

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Understanding the Challenges of Development Teams Employing Kubernetes for AI/ML Containerized Applications

Permalink

<https://escholarship.org/uc/item/8dx6n4fk>

ISBN

9798193222491

Author

Coleman, Cora

Publication Date

2026-10-02

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA SAN DIEGO

Understanding the Challenges of Development Teams Employing
Kubernetes for AI/ML Containerized Applications

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Cora Coleman

Committee in charge:

Professor William G. Griswold, Chair
Professor Philip J. Guo
Professor Nadia Polikarpova
Professor George Porter

Copyright
Cora Coleman, 2026
All rights reserved.

The Dissertation of Cora Coleman is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2026

TABLE OF CONTENTS

Dissertation Approval Page	iii
Table of Contents	iv
List of Figures	vii
List of Tables	viii
Acknowledgements	xii
Vita	xiii
Abstract of the Dissertation	xiv
Chapter 1 Introduction	1
Chapter 2 Motivating Scenarios	4
2.1 Scenario 1: A Data Scientist Running Locally	4
2.2 Scenario 2: A Data Scientist Running in a Kubernetes Environment	5
Chapter 3 Related Work	7
3.1 Developer Perspectives on Tool Design and Usability	10
3.1.1 Surveys	11
3.1.2 Tool Interventions	13
3.1.3 User Studies	15
3.1.4 Takeaways on Tool Design and Usability	17
3.2 Developer Perspectives on Cloud Development	18
3.2.1 General Cloud Development	18
3.2.2 AI/ML Cloud Development	38
3.2.3 Takeaways on Cloud Development	49
3.3 Developer Perspectives on Containerized Development	50
3.3.1 General Containerized Development	50
3.3.2 AI/ML Containerized App Development	60
3.3.3 Takeaways on Containerized Development	85
3.4 Final Remarks on Prior Work	86
Chapter 4 Methods	88
4.1 Top-Down Perspectives	89
4.1.1 Work Partitioning According to Specialization	89
4.1.2 Gulfs of Execution and Evaluation	91
4.1.3 Defining the Relationship Between Directness and Modularity	92
4.2 Bottom-Up Perspective: Straussian Grounded Theory	100
4.3 Participants	100

4.4	Data Collection	105
4.5	Coding Process	105
4.5.1	Inter-Coder Reliability (ICR)	106
Chapter 5	Results	108
5.1	Job Role Versus Task Performance	108
5.2	Dataset Management	112
5.2.1	Summary of Execution Gulfs	113
5.2.2	Summary of Evaluation Gulfs	120
5.2.3	Dataset Management Tools	128
5.3	Debugging	132
5.3.1	Summary of Execution Gulfs	133
5.3.2	Summary of Evaluation Gulfs	141
5.3.3	Debugging Tools	154
5.4	Job Submission	158
5.4.1	Summary of Execution Gulfs	158
5.4.2	Summary of Evaluation Gulfs	180
5.4.3	Job Submission Tools	187
Chapter 6	Discussion	191
6.1	What are the Most Prevalent Challenges?	193
6.1.1	A Divergence From Specialized Tasks	195
6.1.2	Switching Between a Multiplicity of Tools	196
6.1.3	Increased Communication With Others for Help	199
6.2	Who Experiences the Most Challenges?	200
6.3	To What Extent are These Challenges Unique to Containerized Development? ..	201
6.3.1	Challenges in General Containerized Development	204
6.3.2	Challenges in AI/ML Containerized App Development	209
6.4	Recommendations for Future Work on Tool Interventions	223
6.4.1	Solutions From the User Side	227
6.4.2	Solutions From the System Side	229
Chapter 7	Threats to Validity	231
7.0.1	Internal Threats to Qualitative Studies	231
7.0.2	Limits to Generalizability	233
Chapter 8	Conclusion	236
8.1	Takeaways	237
8.2	Contributions	240
Appendix A	Interview Questions	242
A.1	Warm-up	242
A.2	Everyday Process	243
A.3	Development Process	244

A.4	Tools	245
A.5	Typical Scenario	246
A.6	Wrap-up	247
Appendix B	Development Tools	248
B.1	Dataset Management Tools.....	248
B.2	Debugging Tools	251
B.3	Job Submission Tools	253
Appendix C	Codebook.....	255
Bibliography		259

LIST OF FIGURES

Figure 5.1.	Combined tasks and gulfs of execution and evaluation, grouped by job role and development task for 35 participants, including: six project managers, 13 data scientists, four cluster operators, two library developers, and 10 infrastructure developers. Tasks and job roles are further categorized into their interpretations.	110
Figure 5.2.	Overview of the tools used by participants for dataset management tasks. Numbers represent the total amount of participants that use each tool.	129
Figure 5.3.	Breakdown of the tools used by participants with different job roles for dataset management tasks. Colors represent unique tools and numbers represent the total amount of participants that use each tool.	130
Figure 5.4.	Overview of the total number of unique tools used by participants with different job roles for dataset management tasks.	130
Figure 5.5.	Overview of the average number of unique tools used by participants with different job roles for dataset management tasks.	131
Figure 5.6.	Overview of the tools used by participants for debugging tasks. Numbers represent the total amount of participants that use each tool.	155
Figure 5.7.	Breakdown of the tools used by participants with different job roles for debugging tasks. Colors represent unique tools and numbers represent the total amount of participants that use each tool.	156
Figure 5.8.	Overview of the total number of unique tools used by participants with different job roles for debugging tasks.	156
Figure 5.9.	Overview of the average number of unique tools used by participants with different job roles for debugging tasks.	157
Figure 5.10.	Overview of the tools used by participants for job submission tasks. Numbers represent the total amount of participants that use each tool.	188
Figure 5.11.	Breakdown of the tools used by participants with different job roles for job submission tasks. Colors represent unique tools and numbers represent the total amount of participants that use each tool.	189
Figure 5.12.	Overview of the total number of unique tools used by participants with different job roles for job submission tasks.	189
Figure 5.13.	Overview of the average number of unique tools used by participants with different job roles for job submission tasks.	190

LIST OF TABLES

Table 3.1.	List of interview guiding questions for the interviews conducted by Sampath et al. [187].	17
Table 3.2.	The total number of configuration-related commits and their changes in the dataset collected for the large-scale analysis conducted by Zhang et al. [210].	20
Table 3.3.	Interview guidelines and questions for the interviews conducted by Leitner et al. [150].	30
Table 3.4.	Summary of findings within the six themes Lietner et al. identified from their interviews [150].	32
Table 3.5.	General topics and questions from semi-structured interviews with senior IaC developers conducted by Guerriero et al. [133]. For questions Q12–Q17 they defined a Likert scale with five values ranging from “Not Important” to “Extremely Important”.	34
Table 3.6.	Semi-structured interview question framework used to prompt participants for the interviews conducted by Simm et al. [194].	45
Table 3.7.	Further breakdown of the topics identified by Haque et al. within the categories of Docker-related SoF posts in the dataset collected by them for their large-scale analysis [135]. The category of basic concepts is omitted as it did not include any further identified topics.	53
Table 3.8.	Answer choices for 2018 SIG Apps survey question, “I would describe my experience with Kubernetes as: ”, not including “None” or open-ended responses [120]	54
Table 3.9.	Job role, interpretation, and list of primary requirements for users of the Katib framework, built by George et al. [127].	64
Table 3.10.	Cloud service users in scientific computing communities can be classified into three categories depending on their level of ML experience and goals. Garcia et al. built the DEEP framework with consideration for these user differences [125].	67
Table 3.11.	Phases of the MLOps cycle that are supported by the OCDL platform built by Liu et al. [157].	74
Table 3.12.	Job roles of developers that collaborate with each other in the ML application development cycle and their goals. Liu et al. built the OCDL platform with consideration for the collaboration between and goals of these developers [157].	74

Table 4.1.	Summary of the benefits of direct-manipulation systems as defined by Shneiderman [193].	92
Table 4.2.	Summary of the requirements of a system that provides the experience of direct engagement, as defined by Hutchins et al. [142].	98
Table 4.3.	Considering the experience of distance in terms of a developer’s expertise in performing a task, the greater their expertise in performing a task, the less distance they experience.	99
Table 4.4.	Breakdown of the number of participants from each organization, IDs used to represent them, and the average software development experience for each organization.	101
Table 4.5.	Participant job roles from both organizations, total number of participants within each job role, and average software development experience.	101
Table 4.6.	Participant job roles from organization A, total number of participants within each job role, and average software development experience.	102
Table 4.7.	Participant job roles from organization B, total number of participants within each job role, and average software development experience.	102
Table 4.8.	Summary of participants’ job roles and their definitions, along with the higher-level interpretations we applied.	103
Table 4.9.	Summary of participants’ development tasks and their definitions, along with the higher-level interpretations we applied.	103
Table 4.10.	Diaz et al. present several limiting factors of generating an ICR metric in Grounded Theory studies [116].	106
Table 5.1.	Summary of the specific challenges developers experience when performing dataset management, debugging, and job submission tasks for AI/ML containerized app development.	112
Table 6.1.	The percentage of participant responses that indicate they experienced a specific challenge, identified within the gulfs of execution and evaluation for a given task. “Other” represents the percentage participant reports that did not fit into any of these specific challenges.	193
Table 6.2.	Proposed job role correspondences from prior works that focus on similar job roles. “-” refers to the job roles in prior work that do not directly correspond to our own.	203

Table 6.3.	Summary of the specific challenges developers experience when performing dataset management tasks, whether these challenges are addressed in prior work on tool interventions, and recommendations for future work regarding development processes and tools based on our findings.	224
Table 6.4.	Summary of the specific challenges developers experience when performing debugging tasks, whether these challenges are addressed in prior work on tool interventions, and recommendations for future work regarding development processes and tools based on our findings.	225
Table 6.5.	Summary of the specific challenges developers experience when performing job submission tasks, whether these challenges are addressed in prior work on tool interventions, and recommendations for future work regarding development processes and tools based on our findings.	226
Table B.1.	13 unique tools are used by cluster operators to perform dataset management tasks.	248
Table B.2.	11 unique tools are used by data scientists to perform dataset management tasks.	249
Table B.3.	10 unique tools are used by infrastructure developers to perform dataset management tasks.	249
Table B.4.	Four unique tools are used by library developers to perform dataset management tasks.	249
Table B.5.	Nine unique tools are used by project managers to perform dataset management tasks.	250
Table B.6.	12 unique tools are used by cluster operators to perform debugging tasks. ...	251
Table B.7.	15 unique tools are used by data scientists to perform debugging tasks.	251
Table B.8.	20 unique tools are used by infrastructure developers to perform debugging tasks.	252
Table B.9.	Two unique tools are used by library developers to perform debugging tasks.	252
Table B.10.	12 unique tools are used by project managers to perform debugging tasks. .	252
Table B.11.	Six unique tools are used by cluster operators to perform job submission tasks.	253
Table B.12.	14 unique tools are used by data scientists to perform job submission tasks.	253

Table B.13.	12 unique tools are used by infrastructure developers to perform job submission tasks.	254
Table B.14.	Seven unique tools are used by library developers to perform job submission tasks.	254
Table B.15.	14 unique tools are used by project managers to perform job submission tasks.	254
Table C.1.	11 codes related to development processes included in our final codebook. .	255
Table C.2.	14 codes related to the frequency of performing a task (12) and using Kubernetes itself (1) as well as the time spent learning new concepts (1) included in our final codebook.	256
Table C.3.	20 codes related to tools included in our final codebook.	256
Table C.4.	20 codes related to the gulfs of execution (10) and evaluation (10) included in our final codebook.	257
Table C.5.	20 other codes included in our final codebook.	258

ACKNOWLEDGEMENTS

I want to thank my advisor, Bill Griswold for the opportunity to work with him and for helping me achieve clarity in my academic writing as well as gain confidence in presenting my research to the scientific community. I'm grateful for all of the times I met with Bill to discuss my progress and for his selflessness in sharing the wisdom and ideas that those conversations inspired. During graduate school at UCSD, I've achieved all I have because professors and researchers such as Bill allowed me to stand on their shoulders to see the world and make connections from that which they have established.

I also want to thank my committee, Philip Guo, Nadia Polikarpova, and George Porter for their guidance and support. In addition, I want to thank Matt Lepinski and John Doucette for believing in me and suggesting I should apply for my PhD in computer science. I treasure the memory of my initial interview with Bill and how Matt kindly lent me his office for the occasion.

I would also like to thank everyone else who I have had the pleasure of working with at IBM Research. I'm grateful for Nick Mitchell's mentorship during both of my research internships with him at IBM as well as for his continued enthusiasm and support as I finished writing my dissertation. As well, I'm grateful for the IBM PhD Fellowship Award that allowed me to continue working with Nick as my mentor and engage in his creativity to conduct the research presented in this dissertation. I'm grateful to Bill for nominating me for this award and for Nick's support throughout the process – it was a long shot that I couldn't have achieved if they didn't believe in me. The research presented in this dissertation was conducted under an IBM PhD Fellowship Award in the Hybrid Cloud Infrastructure domain.

Most importantly, I want to thank my family and my fiance who have always been there for me. I would not be where I am today without them.

VITA

2018	Bachelor of Arts in Computer Science, New College of Florida
2020	Master of Science in Computer Science, University of California San Diego
2026	Doctor of Philosophy in Computer Science, University of California San Diego

PUBLICATIONS

Diana Arroyo, Paul Castro, Thuan Doan, Nick Mitchell, Sara Kokkila Schumacher, Ed Seabolt, Aleksander Slominski, Ansu Varghese, Lionel Villard, and Cora Coleman, “The Cloud, Like Building and Running Go Binaries,” *2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*, Glasgow, United Kingdom, 1022-1032, 2025.

ABSTRACT OF THE DISSERTATION

Understanding the Challenges of Development Teams Employing
Kubernetes for AI/ML Containerized Applications

by

Cora Coleman

Doctor of Philosophy in Computer Science

University of California San Diego, 2026

Professor William G. Griswold, Chair

As containerized development has grown in popularity, it's been increasingly employed in domains for which it wasn't originally designed, such as AI/ML. Anecdotally, it has been reported that development teams are concerned with the challenges of containerization, despite the promised gains in productivity. There have been few studies focused on understanding developer perspectives in the context of the changes to the software development cycle that arise from employing containers. Furthermore, past studies do not relate the various types of workloads to job roles and the extent of their impact on the specific development tasks performed and tools being employed in containerized development. To address this gap, we

conducted 35 contextual interviews with Kubernetes cluster operators and varying levels of users (i.e., namespace users and managers). Participants were recruited from two distinct research organizations, one industrial and the other academic. Their primary focus was to train AI/ML models or use them in containerized application development. We found that cluster operators and users frequently perform a range of tasks that diverge from their job role expertise. As a consequence, they frequently switch among a multiplicity of tasks, using various tools. This drives them to create custom tool integrations to mitigate the cognitive load of switching between tools (and hence switching between tasks). In addition to a deeper understanding of AI/ML containerized application developer perspectives, our study contributes a set of development process recommendations and tooling implications, in view of prior work on tool interventions.

Chapter 1

Introduction

The widespread appeal of employing containers in software engineering and more recently, scientific research projects is, in part, due to their ability to improve reproducibility and collaboration among teams who may use different tool ecosystems [165]. Container orchestration platforms, the most popular being Kubernetes [84], propose to ease development tasks such as scaling and deploying new versions of applications (i.e., apps), as well as provide monitoring, logging, and debugging support [143]. Despite the increased adoption of containerized development and orchestration platforms, research has yet to provide a deeper understanding of the challenges faced by those employing containerized development in domains for which it wasn't originally anticipated, notably Artificial Intelligence/Machine Learning (AI/ML). Indeed, some of the most popular projects hosted by the open-source Cloud Native Computing Foundation (CNCF) [32] concern AI/ML workloads (e.g., Kubeflow [81]).

Anecdotally, we heard from larger teams, composed of hundreds of software developers, that employing Kubernetes for AI/ML containerized apps has proved to be challenging, despite its purported benefits. Larger teams often have distinct job roles with specialized task responsibilities that are intrinsic to those roles. Therefore, we hypothesized that AI/ML containerized app development, even as orchestrated through Kubernetes, was undermining the separation of responsibilities among roles. However, we did not have a sense as to the nature of how the separation of responsibilities may be undermined or to what extent this may impact a developer's

work. A well-known model of interaction in Human-Computer Interaction (HCI) research, developed as a User Experience (UX) design framework, provides a useful lens for understanding the nature of a person’s struggles when using tools and technology to execute and evaluate their tasks [168, 142].

To gain a deeper understanding of the issues that teams encounter in AI/ML containerized app development, we conducted a contextual interview study of 35 professional developers from two different organizations employing Kubernetes for AI/ML containerized apps. During our interviews, developers shared and explained examples of their work to help us better understand the challenges of executing and evaluating their tasks on a Kubernetes cluster, including walking through and demonstrating their code. In total, 29 developers were interviewed from organization A, a large industrial organization, and 6 developers were interviewed from organization B, a small academic organization. We sought to compare perspectives from these two organizations because, at the time the interviews were conducted, organization A was in the process of employing Kubernetes and setting up development and production clusters, while organization B had established a more mature Kubernetes production cluster to provide resources for research groups internationally.

In support of our hypothesis, we found that developers often had to take responsibility for tasks that diverged from their job roles as infrastructure developers, library developers, cluster operators, data scientists, and project managers. Here, we focus on three tasks (i.e., job submission, debugging, and dataset management), identifying three key issues in AI/ML containerized app development:

1. Job submission tasks require all developers to take on a cluster operator role, thus having to take on tasks, and hence use tools, for which they weren’t trained.
2. Debugging tasks are challenging, in part, due to cluster users having limited visibility of the cluster’s state, which results in additional collaboration between those with varying job role expertise and user privileges.

3. Dataset management tasks come with a steep learning curve, especially for developers adapting their storage from a traditional high-performance computing (HPC) environment to a Kubernetes environment.

From these insights we further contribute a set of development process recommendations and tooling implications to address the challenges that development teams encounter in employing Kubernetes for AI/ML containerized apps.

Chapter 2

Motivating Scenarios

To provide more context for understanding the results of our study, we introduce a scenario that involves a data scientist, illustrating their job role and the set of tasks that they commonly reported performing in our interviews.

2.1 Scenario 1: A Data Scientist Running Locally

Ada Drift is a data scientist by training and performs this job role as part of a larger team of data scientists. As a routine part of her role, she is expected to write Python code, ensure it meets functional and performance goals, publish it in a form her team can use, and document how others can then use it. Ostensibly, her training has prepared her to be able to successfully accomplish these tasks with minimal friction.

When working on her laptop, Ada Drift iterates through an edit-test-debug task cycle that has low friction. Having previously set up her laptop for Python development, including the download of small datasets sufficient for testing, each iteration of development proceeds smoothly: 1) update code, 2) run with `python experiment.py`, and 3) validate the console and file output. She can iterate quickly, without frequently needed to consult with the broader team in order to reach consensus on when and where her tasks should be performed. Broadly speaking, on her laptop these tasks are only troublesome to the extent that writing good code is generally so.

When Ada Drift is ready to share her code with other developers on her team, most of what she has done locally on her machine is sufficient to enable the other data scientists on her team to be successful.

2.2 Scenario 2: A Data Scientist Running in a Kubernetes Environment

Having completed and tested her code locally, Ada Drift shifts her development to the shared Kubernetes cluster. The cluster has GPU resources of kind and quantity unavailable locally. Access to the input data is restricted to machines running behind the private network in which the cluster resides. Because of the way the cluster is managed and how many people are sharing it, Ada Drift's edit-test-debug task cycle is rather different from her local process.

Kubernetes runs code built into container images and it fetches those images from a container registry. To start, Ada Drift writes a Containerfile that describes how to populate the container's image contents. This allows her to build the image. Next, she logs in to the container registry. The registry namespace is shared with her other team members. She communicates with her team to reach a consensus on how to name the image, so as to avoid naming conflicts with other team members. Upon publishing her container image, she then writes a Kubernetes job specification for running the container. In this specification, she provides *secrets* that allow Kubernetes to fetch her image from the private registry. She then provides additional secrets to the job that allow it to fetch data from storage. She then specifies her job's priority. Related to this, she communicates with her team to reach a consensus on how many GPUs she is allowed to request for her job run.

Ada Drift is now ready to submit her job. Before this can be done, a final round of communication with her team may be necessary to ensure that her job does not interfere with any last-minute, critical jobs from other team members. Given permission to run, Ada Drift submits her job to the cluster. Once her job has been submitted, the GPU computing resources

she requested are unavailable to other team members. This leads to a tense, initial period of cluster observation. Perhaps this job has accidentally preempted more important jobs. Perhaps the wrong credentials were used. Perhaps the job description was not updated to point to the new container image, or Ada Drift forgot to publish an updated image.

In the transition from using a laptop to a remote cluster, Ada Drift moved beyond her data scientist training. Although container image and cluster management are tasks that have good tool support, these tools were designed, as is Kubernetes as a whole, for cluster operators who have distinct skills and responsibilities from those of a data scientist. Also of note is that several of these tasks required additional communication with Ada Drift's other team members in order to reach a consensus on a decision or get permission to continue making progress in her task cycle, adding overhead.

Chapter 3

Related Work

There are three key dimensions to the prior work related to our study. One is the distinction between user studies (e.g., surveys, interviews, large-scale analyses) and tool intervention studies. Two is the specificity of these studies regarding AI/ML app development versus a broader application mix, of which AI/ML apps may or may not have been included. Three is the relationship to containerized development in particular, as it is part of the broader category of cloud development. Below, we review work across these three dimensions, revealing that although much is known, there is a notable gap in understanding the challenges faced in AI/ML containerized app development.

There are distinctions between cloud development and containerized development, though they often intersect in software engineering. Cloud development refers to building, deploying, and managing applications using a cloud computing service model, such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), or Function-as-a-Service (FaaS) which is a serverless computing model [131]. Cloud development leverages cloud provider infrastructure (e.g., AWS [19], Azure [21], Google Cloud [38]), APIs, and tools to abstract away hardware management, which can introduce vendor lock-in due to reliance on specific cloud providers [29]. IaaS, PaaS, and FaaS service models have varying degrees of infrastructure transparency, whereby the underlying hardware, operating systems, and server management are minimally, partially, and fully abstracted away from the user, respectively. For example, FaaS developers focus solely

on writing and deploying code, as their cloud provider handles all infrastructure-related tasks.

Cloud development applications can involve various deployment models, including virtual machines (VMs), containers, or serverless functions [28]. An example of a cloud development application includes building a serverless application with AWS Lambda, wherein a developer writes a function to handle image uploads to Amazon Simple Storage Service (S3) without needing to know whether the function runs on one server or on a cluster with multiple servers, how containers are spun up, or how scaling occurs.

Containers are a virtualization technology rather than a standalone cloud service model such as IaaS, PaaS, or FaaS. Containerized development refers to packaging applications and their dependencies into containers [143]. Containers are lightweight, portable units that run consistently across different environments, independent of the underlying infrastructure (e.g., bare metal, on-premises, cloud, hybrid), via technologies such as Docker [34]. Therefore, containerized development does not rely on a cloud provider, as containers can run anywhere with a container runtime. Containers ensure consistency among development, testing, and production environments.

Containerization allows developers to create and deploy applications that employ either a traditional monolithic (i.e., a single-tiered application) or microservices (i.e., the application is broken into smaller, containerized components) architecture [143]. For example, developers can build new cloud-based applications as containerized microservices, breaking a complex application into smaller, specialized services. As well, developers can reconfigure existing applications into containers, or containerized microservices, that use computing resources more efficiently.

Containerized development operates at a lower level of abstraction, similar to IaaS, requiring users to manage container images and orchestration. Containers are frequently used in cloud development to deploy applications on cloud platforms. Serverless applications are a cloud development paradigm and often use containers under the hood (e.g., AWS Lambda [179] uses containers to execute functions) however, the user experience abstracts this layer. An example of

a containerized development application is packaging a microservice in a Docker container or orchestrating containers with Kubernetes.

Another cloud computing service model known as Containers as a Service (CaaS) exists where providers offer managed container orchestration platforms (e.g., Kubernetes services such as the Google Kubernetes Engine) [25]. Regarding infrastructure transparency, CaaS falls between IaaS and PaaS. Users package applications with their dependencies into containers and manage container images, and the provider handles container orchestration, scheduling, and the underlying infrastructure (e.g., servers and VMs). Although CaaS provides more abstraction than IaaS (e.g., hiding details such as OS management), it requires users to handle container-level configurations and offers greater customization and control than PaaS. However, when using containers directly, without CaaS, the level of abstraction is minimal, similar to IaaS. For example, IaaS developers have direct control over virtual machines, OS installation, networking, and storage. Similarly, containerized developers manage container images, orchestration, scheduling, and the underlying infrastructure (e.g., servers and VMs). Our study focuses on developers that use containers directly, without CaaS.

Relevant to both cloud and containerized development, Infrastructure as Code (IaC) is the practice of defining and managing the underlying infrastructure (e.g., servers, networks, storage, clusters) using configuration files rather than doing so manually [14]. Without IaC, managing the computing resources provided by the underlying infrastructure using manual processes is slow, error-prone, and non-repeatable at scale [14, 95, 140].

In the following, we review some prior work broadly related to software developer perspectives on AI/ML containerized app development. First, because tools are such a critical part of containerized development, we review prior work related to developer perspectives on the design and usability of development tools. Next, for context, we review prior work regarding developer perspectives on cloud development. Finally, we review prior work on developer perspectives particularly related to containerized software development.

3.1 Developer Perspectives on Tool Design and Usability

As illustrated in Chapter 2, a AI/ML containerized app development workflow involves writing AI/ML code to run on a cluster, configuring a job submission, submitting a job, monitoring a submitted job, and then analyzing the outcome of a job.

While working on a programming task, developers must maintain a working task context, also known as a problem state, to keep track of task-related information not immediately available in the external environment or in the information associated with performing a task [171, 71]. When relationships between information in the working task context are more obscure, problems become harder to solve and require more steps to reach a solution [171]. For example, steps must be taken to reference related information from prior knowledge and experience, and to understand the task-related information, the desired solution, and any other interrelationships [171, 182].

A developer’s working task context is the main source of interruptions [71]. Seeking out more context leads to both external interruptions and internal, also referred to as self-initiated or self, interruptions. A self-interruption is originated by the subject of the task and an external interruption is motivated by external events [71].

Task switching, also referred to as multitasking or interruption, is defined as the act of starting one task and moving on to another before finishing the first [71]. Context switching is a specific kind of task switching and is defined as switching between different tasks entirely, for example, between different programming task types and development projects [71, 204]. Constantly recreating a working task context contributes to excessive cognitive load [71].

Developers may also switch among a combination of specialized tools with different interface modalities (e.g., CLI, GUI) to accomplish each of these tasks. Many GUI tools have been designed to complement or supplant CLI tools. Anecdotally, it has been reported that novice developers find CLIs to be more challenging to use. To back up these reports, prior work has been conducted to understand the tradeoffs between CLIs and GUIs. Below, we discuss one survey [206], tool intervention [203], and user study [187] related to this subject that provide

relevant context for understanding the results of our study.

3.1.1 Surveys

Survey of system administrators reveals that experts prefer firewall tools with CLIs over GUIs.

Voronkov et al. conducted a survey of system administrators (i.e., admins) to understand: 1) their preferences regarding user interfaces (UIs) for firewall tools, such as CLIs, GUIs, and APIs, as well as 2) which interfaces are more suitable for certain tasks [206]. They note that although prior work has found CLIs to be the main tool interface modality used by system admins, they want to gain a deeper insight into the reasoning behind system admins that use various firewall tool modalities.

In 2018, they conducted a survey over six weeks from April to June. Participants for the survey were recruited via various channels, including system admins' forums (i.e., "Sysadmin" subreddit and SysAdmins.ru) and system admins' mailing lists. In total they received responses from 300 system admins. They used skip logic [65] to formulate the survey, which consisted of up to 14 questions, four of which were formatted as free-response. The survey was composed of two parts, with the first asking participants about different interactions with firewalls (i.e., the average time they spend working with firewalls, which firewall interface they mainly work with and which they prefer, which tasks are easier with which firewall interface, as well as the strengths and limitations of those interfaces).

They analyzed the survey data using a content analysis approach. Two authors worked independently and coded participants' responses to the free-response questions using an initial open coding approach [186]. They used NVivo [55] to conduct manual coding as well as organize and analyze the data. Coding was performed in two rounds, before and after the final codebook was generated. After the authors completed the first round of coding, they discussed their codes together to resolve any differences and reach a consensus. As multiple people coded the data, they used Cohen's Kappa (κ) [105] to generate an inter-rater reliability value and found their value to

be 0.79 for the final codebook, which indicates an excellent agreement among coders [123].

They coded and further grouped the opinions of the system admins according to two principles: 1) the type of interface (i.e., CLI, GUI, API, or other) and 2) the type of comment (i.e., positive, negative, or neutral). They further grouped the benefits and limitations of these interfaces based on Jakob Nielsen's 10 usability heuristics [167]. The codes that did not concern usability (i.e., security and reliability) were further grouped according to the ISO/IEC 25010, a standard model for software quality [2]. All remaining codes were further grouped into an "Other" category. Nielsen is one of two UX pioneers that founded the organization Nielsen Norman Group (NN/g) [54], the other being Don Norman who is one of the contributors to the well-known model of interaction in HCI research that was used to group the findings in our study into two categories known as the gulfs of execution and evaluation [142].

They presented findings regarding the benefits and limitations of CLI and GUI firewall interfaces, along with which tasks they are most suitable for. Regarding CLI benefits, they found the top three most reported to be: 1) the flexibility and efficiency (33.2%), noting that they give users the ability to perform automation, 2) the functional suitability (19.4%), noting that they provide superior functionality, and 3) the usability (9.4%), noting that they give users full control. Regarding CLI limitations, they found the top three most reported to be: 1) the match between the system and the real world (25.6%), noting that they have a steep learning curve, 2) the usability (25.6%), noting that they are not easy to use, and 3) the recognition rather than recall (11.7%), noting that they are less intuitive to use.

Regarding GUI benefits, they found the top three most reported to be: 1) the usability (40.3%), noting that visual representations of data provide a better understanding, 2) the functional suitability (20.5%), noting that they are easy to use, and 3) the recognition rather than recall (14.2%), noting that they are good for occasional use as they are easy to navigate. Regarding GUI limitations, they found the top three most reported to be: 1) the flexibility and efficiency of use (30.8%), noting user inefficiency and a lack of automation options, 2) the functional suitability (13.8%), noting the reduced functionality provided, and 3) the match between the

system and real world (9.4%), noting that the additional layer of abstraction may prevent the user from gaining a deeper understanding.

They revealed which tasks CLI and GUI firewall interfaces are most suitable for. They found that most system admins (70%) work primarily with firewall GUIs and most (60%) also prefer GUIs as a main tool, especially for monitoring tasks for which they provide better data visualizations. However, they found that expert system admins did not prefer GUIs over CLIs because they lack automation and generally have inefficient UIs. For example, CLIs are relied on for bulk changes and non-standard tasks. They recommend design improvements for CLIs, which included assistance in writing rules for novice system admins because CLIs have a steep learning curve. They also recommend design improvements for GUIs, which included the ability to allow users to create hotkey combinations for the most popular actions.

3.1.2 Tool Interventions

Bespoke system automatically creates custom GUIs from CLI workflows to support novice developers.

Vaithilingam et al. built Bespoke, a system to automatically synthesize custom GUIs by documenting user interactions, and conducted a user study to further understand how to make command-line applications more usable [203]. They note that although command-line applications pose usability challenges for novices, such as recognition over recall [96], it is challenging to design a GUI to replace the many, complex commands provided in a CLI. Furthermore, they note that manually developing a custom GUI requires extensive development effort, including continuous maintenance of the UI as user needs change. We reflect on how tools can be built to support users as they gain expertise and their needs change in Chapter 6.

Bespoke is a cross-platform Electron [107] desktop app that enables users to run command-line applications in terminals and view GUIs synthesized from those runs. Bespoke users can create two types of command cells: 1) a Markdown documentation cell [132] and 2) a command cell where they can run a command-line application and have Bespoke synthesize a

custom GUI. Bespoke synthesizes a GUI in three-steps: 1) parsing its command-line arguments, 2) inferring the types and value ranges for those arguments, and 3) mapping those types and value ranges to standard GUI elements. Bespoke uses a programming-by-demonstration approach such that each time the user runs a command, it gets added to the cell history, and its synthesized GUI updates accordingly. Users can manually remove history entries to view their effects on the GUI's synthesis and override any GUI elements to change the types and value ranges.

They conducted a user study to understand: 1) what kinds of GUIs users create with Bespoke as well as 2) the benefits and limitations of Bespoke and the GUIs created. They recruited six graduate students with command-line application experience. These participants had jobs roles as engineering interns, scientific researchers, and teaching assistants. Participants were first given a tutorial of Bespoke. Then, participants were asked to think of possible uses for Bespoke in their work.

They encouraged participants to think about workflows that involved multiple parts, interactive exploration, and manual inspection of outputs. Then, the participants used Bespoke to run the appropriate commands on their machines to synthesize and customize their GUIs with the goal of supporting their personally motivated workflow. They encouraged participants to think aloud about the perceived benefits and limitations of Bespoke. These individual sessions lasted one and a half hours wherein participants used Bespoke to create their own GUIs in a variety of domains (i.e., cloud computing management, machine learning prototyping, lecture video transcription, integrated circuit design, remote code deployment, and gaming server management).

They found that command-line applications offer flexible workflows that allow experts to easily adjust parameters, link outputs through pipes and files, and create scripts to combine multiple tools. However, they note that they pose significant usability challenges for novices, particularly due to their reliance on recall over recognition. For example, they note that users must remember and input precise, complex syntax of parameters instead of selecting options from a user-friendly GUI. One of Nielsen's aforementioned 10 usability heuristics is to promote

recognition over recall in UI design [96]. Regarding this limitation, they recommend that CLIs include more demonstrative documentation and visualizations to increase system feedback, such as for status and progress indication.

They found that participants tend to prefer CLIs for their completeness of functionality as they provide many options for automating simple, repetitive tasks and generally have a more efficient UI compared to GUIs. They found that people prefer GUIs for more specific tasks that benefit from data visualizations, such as monitoring and parameter tuning. They conclude that developers tend to employ a variety of CLIs and GUIs to optimize their work in regards to these preferences and that these tendencies remain in a cloud computing context. They found the main limitations of Bespoke, including how it does not handle software environment differences among computers, and recommend that containers could alleviate this issue and remove some of the obstacles of converting CLIs to GUIs.

3.1.3 User Studies

Mixed-methods user study evaluates the accessibility of CLIs for developers that use screen readers and finds universally experienced limitations, with a steeper learning curve for novices.

Sampath et al. conducted interviews and user study to evaluate the accessibility of CLIs for developers with visual impairments who use screen readers [187]. They note that although CLIs are thought to be more accessible compared to GUIs due to their text and keyboard interface, these features do not inherently provide accessibility and their features only meet a subset of the WCAG criteria [99].

They conducted semi-structured interviews with six developers who use CLIs and screen readers. They included participants that self-identified as developers that use screen readers and CLIs as part of their development workflow. All but one of the participants had some form of visual impairment.

During the interviews they asked participants about their workspace and assistive technology usage, development practices, the role of CLIs in their developmental workflow, the issues

with using CLIs, and their ideal experience of using CLIs with screen readers. Four interviews were conducted remotely over Google Meet [41] and two interviews were conducted over the phone. The interviews lasted 90 minutes. They recorded and transcribed the interviews for analysis.

They also conducted a user study and second round of interviews to understand the accessibility issues faced by developers with visual impairments when using CLIs as well as further refine and confirm the findings from their initial interviews. Six participants were recruited using the same inclusion criteria as the initial interviews. All participants reported having some form of visual impairment and self-identified as legally blind.

For the user study, six developers remotely completed four tasks over two weeks, using the gcloud CLI [35] in the Google Cloud Platform [38]. These tasks included: 1) installing and setting up the CLI, 2) creating a VM in the cloud, 3) recognizing text from a given image using a public ML service, and 4) deploying a Docker container to the cloud infrastructure. All participants used the JAWS screen reader on a Windows machine to complete these tasks.

They conducted the user study in three parts, the first being an onboarding interview to explain the tasks, share credentials and access to the cloud platform, as well as conduct a brief interview to understand their development workflows. In the second part of the user study, participants worked on the tasks over two week and recorded entries of their experiences, guided by a provided structure. The provided structure consisted of five guiding questions, shown in Table 3.1. Participants recorded their perceived workload in completing each task using three modified NASA TLX [136] 7-point Likert scales [153] for: 1) performance (i.e. success), 2) effort, and 3) frustration. They also had participants estimate how long each task would take them to complete as well as keep track of the time they started and stopped working on each task.

In the third part of the user study, they conducted a final round of interviews with participants to reflect on their experiences, including the difference between their estimated and actual task completion times. The transcripts from all interviews were coded using a Grounded Theory approach [101]. They used NVivo to maintain the codes. They found that although

Table 3.1. List of interview guiding questions for the interviews conducted by Sampath et al. [187].

Interview Guiding Questions
1. Give a brief overview of the steps you used to perform this task.
2. Did you complete this task?
3. How long did you spend on this task?
4. What accessibility or usability issues did you encounter on this task?
5. What resources or documentation did you use for this task?

the task success rates of the user study were high, effort and frustration were also high. They found that this is also reflected in the estimated and actual time differences. They conclude that although the high success rate supports the perception that CLIs are accessible, high effort and frustration demonstrate that the UX of using a CLI with a screen reader is challenging.

They revealed some reasons for inefficiency that are universally experienced, though with a steeper learning curve for novice developers. For example, they found that the quality of error messages ranged from being difficult for screen readers to verbalize, to not being actionable by users. They recommend design improvements to CLIs, which included documenting the output structure for each command to better inform users ahead of running the command, so that they can more effectively parse and comprehend the output. They also recommend providing some form of status and progress indication for commands to increase system feedback, and to ensure that error messages are more actionable.

3.1.4 Takeaways on Tool Design and Usability

In general, these prior works found that experts prefer CLIs for several reasons, including because they provide options for automating repetitive tasks and have a more efficient UI compared to GUIs [206, 203]. They also found that novices experience significant usability challenges when using CLIs due to a reliance on recall over recognition [203]. For developers of all levels of experience, they found that the quality of CLI error messages varied, with some not being actionable by users [187].

In general, these prior works found that GUIs are preferred for tasks that benefit from data

visualizations, such as monitoring and parameter tuning [206, 203]. Therefore, although tools may be designed for developers with varying levels of experience, in practice, developers choose the modalities based on their task context rather than their level of experience. To the extent that these results generalize to our study’s domain of AI/ML containerized app development, we’d expect to find similar challenges. Furthermore, our study contributes to understanding how these challenges are compounded when developers frequently switch among using multiple tools that have different modalities.

3.2 Developer Perspectives on Cloud Development

3.2.1 General Cloud Development

There are interesting research studies related to general cloud development that provide relevant context for understanding the results of our study, such as large-scale analyses. Below, we discuss two large-scale analyses [210, 207] and three user studies [104, 150, 133].

3.2.1.1 Large-Scale Analyses

Large-scale GitHub repository analysis of configuration-related commits finds a lack of automation and standard practices among developers using mature cloud systems.

Configuration files, such as Yet Another Markup Language (YAML) configuration files, are one of the main components used in containerized development. Therefore, understanding the particular challenges that configuration creates is relevant to the results of our study.

Zhang et al. conducted a large-scale GitHub repository analysis of open-source cloud systems to understand: 1) how and why developers evolve configuration design and implementation, as well as 2) the challenges that they face in doing so, particularly regarding the frequency and impact of misconfigurations [210]. They note that the goal of a configuration is for users (e.g., cluster operators, system admins, DevOps engineers) to employ different parameter values such that they can deploy software in various environments, support varying workloads, and fulfill new user requirements. They emphasize that configuration parameters are frequently changed in

cloud systems and as a result, are prone to misconfigurations (e.g., incorrect parameter values) that cause system failures, service outages, and downtime. They point out that although prior work has focused on detecting misconfigurations and evaluating their impact to improve system-level protections, these improvements do not address the challenges of configuration design and implementation.

They conducted a large-scale GitHub repository analysis of four open-source cloud systems (i.e., HDFS [44], HBase [43], Spark [66], and Cassandra [26]) to understand: 1) configuration design and implementation practices, as well as 2) identify the challenges developers face, particularly to inform future automated solutions that address misconfigurations. From the repositories of these systems, they identified configuration-related commits over two and a half years, from June 2017 to December 2019. They identified commits that revise or refine initial configuration design and implementation decisions as a result of misconfigurations.

To create the dataset, they wrote scripts to automate the analysis of commit messages and diffs, filter out irrelevant commits, as well as select likely configuration-related commits. Then, they manually reviewed each commit and its related issue. Initially, they manually conducted a formative study using hundreds of commit messages and identified three strings that frequently occur in configuration-related commits (i.e., “config”, “parameter”, and “option”). These strings matched 525 commits in all four cloud systems. Then, they manually filtered out any irrelevant commits, resulting in 384 configuration-related commits. They then analyzed the commit diffs and found 794 more configuration-related commits.

In total, the final dataset consisted of 1178 configuration-related commits. Each commit was associated with either a JIRA or GitHub issue, or a pull request link that provided further context about the change and any discussions among developers. They found detailed context about changes in these discussions.

To further analyze the commit diffs, they evaluated whether they contain modifications of how parameters are defined, loaded, used, or described. To accomplish diff analysis at scale, they conducted a text search of configuration metadata, including the configuration interface (i.e.,

how configurations are defined and loaded) and default configuration file. Table 3.2 summarizes the commits found from conducting this text search.

Table 3.2. The total number of configuration-related commits and their changes in the dataset collected for the large-scale analysis conducted by Zhang et al. [210].

Changes Made in Configuration-Related Commits	Total (Number of Commits)
Default configuration files or parameter descriptions	272
Code containing getter or setter API methods for retrieving and assigning parameter values	457
Parameter values	31
Exception or log messages without modifying any other code	34

To categorize the final dataset of configuration-related commits, they presented a taxonomy of configuration design and implementation changes in cloud systems along three dimensions: 1) interface (i.e., why and how developers change the configuration parameters, as well as the default values and constraints used), 2) usage (i.e., how developers change and improve parameter value checking, error-handling, and uses), and 3) documentation (i.e., how developers improve configuration documentation). Using this taxonomy, two authors categorized each commit based on the changes made to the original configuration design and implementation.

As two authors categorized the dataset, they held regular meetings together and with others to ensure consistency. The two coders met to compare their categorizations and if they diverged, a third person was consulted for additional discussion until consensus was reached. Furthermore, twice a week the two coders met with a fourth person to review their categorization of 15% of the commits they reviewed during the week.

Out of the three dimensions presented in the taxonomy to categorize the commits, they found a majority (62.1%) were related to configuration interface changes. Regarding interface commits, they focus on analyzing the flexibility of user-facing configuration changes (e.g., parameterization and removing parameters) and default values. Regarding parameterization, they found that there is a need for configuration techniques and tools to automatically choose and test parameter values. They recommend that use cases and workloads in which parameterization is called for be identified and documented, as well as should suggest configuration values designed for specific use cases.

Regarding default values, they found that there is a need for identifying workloads or use cases for which default values are unsuitable. They conclude that such automatic workload identification approaches regarding parameterization and default values can help developers make decisions (i.e., about which constant values need to be parameterized, understand when default values will lead to system failures, and more thoroughly test and benchmark default values). We discuss the importance of default values for AI/ML containerized workloads in Chapter 5.3.2.1.

They found the second highest (30.5%) were related to configuration usage changes. Regarding usage commits, they focus on changes to parameter checking, error handling, and the use of parameters. They found that even among these four mature, production-quality cloud systems, these development processes are often not designed or implemented in a standard way. They found the least (9.7%) were related to configuration document changes.

They note that in general, the accuracy and effectiveness of technical documentation is a long-standing problem in software engineering. They found that there is a need for techniques to maintain and improve configuration documentation. They made a couple of recommendations: 1) documentation-as-code techniques to eliminate inconsistencies and 2) automating document generation for parameter comments.

Large-scale analysis of serverless-related Stack Overflow posts finds that serverless application development presents complex challenges for developers, despite it's abstraction of operational tasks.

Serverless development frequently makes use of containers. Therefore, understanding the particular challenges that serverless application development creates is relevant to the results of our study.

Wen et al. conducted a large-scale analysis of Stack Overflow (SoF) posts to identify the challenges of designing, implementing, and deploying serverless applications [207]. They note that to use serverless computing, developers do not need cloud computing expertise as it abstracts away most operational tasks to simplify cloud application development. For example, they

note that serverless platforms provide VMs or containers to host functions, and automatically manage scalability, fault tolerance, and other runtime issues. They clarify that this is different from traditional cloud computing, where developers often need to continuously manage and run servers, as well as pay for them. When using serverless computing, they note that developers write source code in the form of functions, along with a specific event trigger (e.g., adding a file into cloud storage). Aside from writing a function, they note that developers package and upload their code, as well as specify environment details (e.g., language runtime, memory).

They point out that although prior work has addressed the limitations of serverless platforms (e.g., high startup time and restricted communication bandwidth) and presented solutions to alleviate them, the specific development tasks that challenge serverless application developers are not yet understood (e.g., configuring resources and environments, networking). They emphasize that serverless platforms lack support tools, which results in more challenging debugging and testing tasks. They note that serverless application developers frequently seek technical help on SoF and that these posts could provide insight into understanding the challenges they face. They emphasize that such an understanding can help developers avoid common issues and inform researchers as well as cloud providers on what to focus on improving, regarding the tools, platforms, and development practices used in serverless application development.

They conducted a large-scale analysis of serverless-related SoF posts to understand the popularity of serverless computing practices and the challenges of developing serverless applications. To create the dataset, they first downloaded the entire SoF dataset from the official Stack Exchange Data Dump [106] in December 2020. The downloaded dataset included a total of 20,511,138 posts from July 2008 to December 2020. Guided by prior work [86, 102, 159], they constructed an initial list of tags (i.e., “serverless” and “faas”) related to serverless computing and used these to filter the dataset to retain only the posts that had at least one of the tags. Then, they added more tags to the initial list by extracting the tags from these posts if they matched at least one other tag in the dataset.

Guided by prior work [86], they further refined the tag list and retained only the tags they

found to be significantly relevant by using two heuristics μ and ν to measure the significance and relevance of a tag, respectively. They define a tag as significantly relevant to serverless computing if its μ and ν are higher than specific thresholds. They employed the lowest thresholds used in prior work [86, 159] and only retained the tags whose μ is higher than 0.005 and whose ν is higher than 0.05. Lastly, two authors jointly examined each tag in the refined list. The final tag set consisted of 13 tags. They were able to extract 22,731 relevant posts that each had at least one of those 13 tags.

To analyze the popularity trend of the topic of serverless computing, they were guided by prior work [75, 77, 86, 88, 102, 184, 208, 209] to calculate the number of posts and users related to serverless computing per year for each year from 2015 to 2020. They found that both the number of questions and users have increased regarding serverless-related posts. They found that in 2016, the number of questions and users regarding serverless-related posts increased by more than 340% compared to the previous year. They compared the increase of popularity to the growth rate of the overall number of posts on SoF as a baseline. They found that although the growth rate of SoF questions and users from 2015 to 2020 ranges from 6% to 22%, the growth rate of questions and users regarding serverless-related posts ranges from 47% to 380%. They conclude that serverless is becoming increasingly popular with developers.

To measure the difficulty level of answering questions related to serverless computing, they were guided by prior work [74, 75, 77, 86, 102, 184, 208, 209] to adopt two metrics: the percentage of questions with no accepted answer and the median response time needed to receive an accepted answer. Regarding the first metric, they used the proportion test [166]. Regarding the second metric, they calculated the median value of response time to receive an accepted answer for both serverless-related questions and SoF questions overall.

They also calculated the average difficulty level of all SoF questions as a baseline and found that 48.3% have no accepted answers and an overall median response time of 35 minutes. They found that serverless-related questions had the highest percentage of questions with no accepted answer (61.5%) and median response time (190 minutes). They conclude that serverless-

related questions are more challenging to answer than average SoF questions and questions related to software development (e.g., web development, concurrency, mobile).

To analyze the specific challenges that developers are faced with, they constructed a taxonomy of challenges. Guided by prior work [103, 159], they created the taxonomy using only questions that had an accepted answer, which resulted in a dataset of 8,751 serverless-related questions. Guided by prior work [72], they randomly selected a sample of questions from this dataset, which resulted in a dataset of 619 questions. Then, they randomly sampled 70% of this dataset to construct an initial taxonomy of challenges.

Working together to analyze the posts, two authors employed an open coding method to inductively create the categories of the taxonomy in a bottom-up approach. A third person acted as an arbitrator to help discuss and resolve any disagreements in categorizing questions to reach a consensus. As multiple authors coded the data, they conducted a reliability analysis to ensure consistency. The remaining 30% of the dataset questions were independently categorized by the two coders. They used Cohen's Kappa (κ) to calculate the inter-rater agreement of the independently categorized data. They found that the value of their inter-rater agreement is 0.836, and note that this indicates an almost perfect agreement and consistent coding procedure [148]. The resulting taxonomy consisted of seven broad categories, identified from serverless-related SoF questions: 1) general questions (7.9%), 2) application design (16.3%), 3) environment configuration (5.9%), 4) low-code development (19.6%), 5) application implementation (35.4%), 6) application deployment (4.4%), and 7) non-functional requirements (10.4%).

Regarding general questions, they found that a majority (70.0%) of the challenges in this category fall into the sub-category of conceptual questions. They found that developers frequently experience challenges related to fundamental concepts. They recommend that developers invest time in understanding the underlying working mechanisms, proprietary terms, and platform-specific restrictions of serverless computing. They also recommend that cloud providers create more organized documentation to assist developers in achieving such understanding.

Regarding application design, the majority (67.0%) of these challenges fall into the

sub-category of functionality design. They recommend that cloud providers create practical function templates and instructions to help developers efficiently construct functions. They also found that a majority of functionality design questions (90%) are regarding integrating multiple services. They note that this is frequently done in serverless application development and is complex as developers need to design functions for multiple cloud services. They recommend that researchers and cloud providers summarize a standard design specification to help developers better construct their functions when combining multiple cloud services. For example, they recommend that cloud providers maintain a function store that allows developers to reuse existing functions for their serverless applications.

Regarding environment configuration, they found that a majority (40.5%) of these challenges fall into the sub-category of version compatibility, specifically regarding how to install appropriate versions of tools. Regarding low-code development, they found that a majority (56.5%) of these challenges fall into the sub-category of provider/function configuration. They identified a need for tools that support code validation to help developers debug configuration errors. They found that developers lack guidance from a comprehensive configuration specification and recommend that cloud providers create more instructions and examples for different configuration requirements, particularly to support event and permission settings. They recommend that framework developers abstract away the underlying differences of serverless platforms to provide a standard such that developers do not need to spend additional time reading documentation for multiple platforms.

Regarding application implementation, they found that package integration and functionality implementation are the top two sub-categories, accounting for 30.8% and 21.4% of these challenges, respectively. Regarding package integration, they recommend that cloud providers improve logging and monitoring services to help developers debug function timeout limits that are difficult to detect. Regarding functionality implementation, they compared serverless-related posts to average SoF posts and found that these challenges account for 66.7% and 33.3% of posts, respectively. They conclude that this demonstrates how serverless computing increases the

development complexity, despite its abstraction of most operational tasks. They recommend that cloud providers offer better development and debugging tools to assist developers.

Regarding application deployment, they found that developers have diverse deployment requirements and recommend that cloud providers offer custom deployment templates for developers to use that clarify necessary and optional information. Regarding non-functional properties, they found that networking and performance are the top two sub-categories, accounting for a total of 53% of these challenges. They found that networking-related challenges are related to VPC settings. They found that performance-related challenges are broad and when compared to the percentage of challenges captured by the application implication taxonomy category, they conclude that developers ask more questions regarding the correctness of their programs than its performance.

3.2.1.2 User Studies

Mixed-methods user study reveals the complexity of DevOps communication, how developers use multiple tools, as well as the challenges of using metrics to troubleshoot and maintain cloud applications.

DevOps is a main component of cloud development. Therefore, understanding the particular challenges that DevOps creates is relevant to the results of our study.

Cito et al. conducted interviews and a survey to identify: 1) how applications are developed on top of cloud environments as well as 2) their impact on development practices, particularly regarding the tools used [104]. They note that although prior work focuses on the management of cloud infrastructures and its application to HPC, there is a lack of user-driven research on cloud computing. They conducted a user study based on the techniques found in Grounded Theory [138]. They were guided by prior work [93] in using a mixed-methodology that involved three parts of data collection and analysis: an initial set of interviews, followed by a survey and a second round of interviews.

In the first part of the user study, they conducted interviews to understand how the development and operation of applications change in a cloud environment and what kind of

tools and data developers use for developing cloud software. They conducted semi-structured interviews with developers that had professional experience deploying software in the cloud. Interview participants were recruited from the author's industry partners and personal network. They recruited participants from organizations of varying size, including both small, with 1–100 employees, and large, with more than 100 employees. They recruited 16 participants from 5 different companies that had 3-23 years of professional development experience.

They defined a set of 23 interview questions based on their research questions. One author conducted the interviews, either face-to-face and on-site with the participant or remotely over Skype [199] lasted 30-60 minutes, and were recorded to create transcripts for analysis. Two authors analyzed the transcripts using an open coding technique to iteratively code and categorize participants' statements, resulting in a set of findings. They note that all findings are supported by the statements of multiple participants.

In the second part of the user study, they conducted a survey to further support their findings. The survey had 32 questions that were mostly inspired by the initial findings from their interviews. The questions were formatted in a five-point Likert scale that asked participants to state their agreement and included three free-response questions regarding the overall differences in cloud versus non-cloud development, limitations of cloud development, and tooling used in cloud development.

They targeted the survey to developers with cloud experience by gathering GitHub profiles of the users that followed a number of popular cloud platform repositories (i.e., Amazon Web Services [19], Heroku [45], Google Cloud Platform, Cloud Foundry [30], and OpenStack [57]). They contacted the users that had a public email address in their profile with a description of and link to their online survey. They encouraged participation in the survey by offering a raffle to all participants for the chance to win one of two \$50 Amazon gift cards. In total, they contacted 2,000 GitHub users and received 294 responses. Regarding the respondents' job roles, 71% stated they worked as a software developer, 22% as team lead or product owner, 4% as operations engineer, and 3% listed software architect, researcher, or CTO. They found that respondents had

an average professional development experience of nine years.

In the third part of the user study, they identified four topics of interest from the survey: fault localization, monitoring and performance troubleshooting, cost of operation, as well as design for scalability. To gain more insight into these topics, they defined a set of 27 questions and conducted another round of semi-structured interviews that followed the same protocol as the first round of interviews. Interview participants were recruited from the author's personal network. They recruited nine participants from different organizations that had an average of eight years of professional development experience. The second round of interviews were generally shorter in length, lasting 30-45 minutes. Based on the topics and categories identified in the previous two parts, they used open coding to categorize statements and gather insights into the difference between cloud and non-cloud development.

They report on how cloud computing has affected application development and operations, as well as on the main drivers for these changes. Regarding troubleshooting and maintenance, they conclude that these tasks have seen the most change in cloud development. They found that infrastructure maintenance has become a task that involves adapting infrastructure code files rather than managing live server instances, as servers are no longer viewed as durable entities. Overall, they found that cloud instance issues are frequently not accessible or have already been discarded, therefore searching for logs in production is not possible. They found that instead, developers must anticipate and define relevant logs and collect them in a central repository ahead of deployment. They note that although these practices are not exclusive to the cloud, instance volatility requires their usage in cloud computing.

Regarding DevOps communication, they note that cloud computing goes along with a DevOps style of communication and leads to frequent collaboration between developers and operations engineers. They found that although 12 of the interview participants validate this notion and agree that there is frequent collaboration between developers and operations engineers, the remaining participants reported that dedicated development and operations teams work more independently. They found that even the organizations that identify as following DevOps have

separate teams of engineers that solely implement functional features and those that mostly develop infrastructure code.

They report on how tooling has evolved in the cloud, what kind of metrics are available to use, and how developers use these metrics. In the survey, they asked what tools developers use specifically for cloud development. Regarding cloud tooling, they found that 124 survey respondents (42%) reported using multiple tools. They found that a majority of tools are used for performance tasks (57%), followed by log management and analytics tasks (39%). They found that developers rely on their experience and intuition to solve problems, rather than using existing metrics.

They note that although many different cloud metrics are available, most monitoring tools are built to be used by operations teams, rather than developers without operations expertise. As a result, they conclude that the existing metrics are difficult to leverage for developers. They recommend that these existing tools provide better APIs for data extraction and integration in the development process. They suggest that possible solutions could include integrating such data using IDE plugins or issue tracking systems.

Mixed-methods user study reveals FaaS and serverless application developers struggle with debugging and testing as well as experience steep learning curves from new conceptual models.

FaaS and serverless development frequently make use of containers, as discussed at the beginning of this Chapter. Therefore, understanding the particular challenges that FaaS and serverless application development create is relevant to the results of our study.

Leitner et al. conducted interviews, a literature review, and survey to identify: 1) the types of applications FaaS computing is used for, 2) their architectural styles and development practices, as well as 3) the tradeoffs of using FaaS and serverless computing, including the reality of how their benefits create new challenges for developers [150]. They note that the most widely known implementation of the serverless model are FaaS offerings (i.e., AWS Lambda, Azure Functions [23], IBM's [11] and Google's Cloud [39] functions). They note that application

developers using FaaS benefit from simplified deployments, reduced operations tasks, and “pay-as-you-go” costs, as their service providers are able to efficiently support many users and functions with relatively few resources. However, they note that these benefits come with other challenges and limitations. For example, they note that AWS Lambda and other FaaS services are intended to host small microservices and as such, composing them into, or decomposing from, larger applications is a challenge. They emphasize that a deeper understanding of FaaS and serverless development can inform FaaS providers on what to focus on regarding improving offerings, tools, and development processes.

They conducted interviews to understand the lessons learned, success factors, and best practices of developers employing FaaS and serverless computing. They recruited participants that had production experience with at least one FaaS technology (i.e., AWS Lambda, Azure Functions). They conducted semi-structured interviews, using the guidelines shown in Table 3.3. The interviews were conducted by multiple authors remotely over Skype or Google Hangouts [40], lasted 30-60 minutes, and were recorded to create transcripts for analysis.

Table 3.3. Interview guidelines and questions for the interviews conducted by Leitner et al. [150].

Interview Guidelines	Questions
Introduction	• Have you used serverless in the past? How often? • How important was serverless in this process? Was it periphery or core technology in the project? • Can you explain the rationale for choosing serverless, any alternatives, and why weren’t they chosen? • What services have you been using?
Architecting with Serverless	• How do you choose what goes into a function? How do you split business logic between functions? • Which features are particularly suitable to be implemented in functions? Which are not? • Is there a difference between architecting serverless apps to “regular” microservices? • Typically, how large are functions in your apps? How many do you have? • Typically, are your services event-driven, or otherwise (i.e., accessed over a REST interface)? • Are components standalone, or are functions tightly coupled (i.e., to other functions/parts of the app)?
Advantages and Disadvantages	• What are the major advantages in serverless besides not having to manage servers? • Specifically, what about: reusability, scalability/elasticity, costs, any others? • What are the major disadvantages? • Specifically, what about: testing functions, keeping track of app control flow, managing state across functions, tool support, integration with processes and frameworks (i.e., CI/CD), any others?
Wrap-up	• Do you see serverless as a fad, or will be here to stay? • Is there something that conceptually needs to change before serverless will be ready for prime-time?

To analyze the transcripts, two authors independently reviewed each transcripts and produced a set of codes using the open coding methodology. Then, the initial codes were reevaluated using codes produced from conducting a literature review of informal publications

(i.e., articles). They created a dataset of 50 articles for the literature review, primarily focuses on articles, blogs, and discussions in comment threads on the Hacker News [42] portal from May 2015 to October 2017. They used the Hacker News search engine Algolia [15] to find sources using four search terms (i.e., “serverless”, “aws lambda”, “azure functions”, “openwhisk”) generated in an iterative and exploratory manner. They included sources that described reference architectures, case studies, or experience reports, and excluded any tool announcements or outright marketing communications.

After the initial open coding of the interview transcripts, one author read each article along with any comments it had, excluding those that were down voted and unrelated, and reevaluated the initial codes with any new codes that emerged from the sources. Next, the two authors that initially coded the data discussed and resolved any differences in their coding to reach a consensus. Then, together they conducted axial coding, which led to an initial theory regarding FaaS usage in practice that was discussed with all authors. The results of axial coding were used to develop the survey questions.

They conducted a survey to to validate their qualitative findings on a larger sample of participants. They distributed an anonymous web survey using the survey tool Typeform [70]. The survey consisted of 40 questions in seven categories: demographics (5), terminology (1), application architecture (14), FaaS development practices and patterns (10), the FaaS mental model (5), advantages and challenges when using FaaS (3), and the future of FaaS (2). The questions were a mix of multiple-choice questions, numeric range choices, Likert scales, boolean yes/no questions, and free response.

They used a convenience sampling methodology [7] and distributed the survey through different channels. The survey was available from January to February 2018, and received 182 responses. They found that a majority of respondents reported to be working in software developer or architect job roles. They found that they also received responses from product owners, IT managers, site reliability engineers, DevOps engineers, researchers, and executives.

They found six themes in the results of their interviews and survey: 1) mental models, 2)

types of serverless applications, 3) application patterns, 4) development languages and patterns, 5) advantages and challenges, as well as 6) deployment costs. They define that the term “serverless” is used represent the general cloud service development model, and “FaaS” is used to specifically refer to cloud services (i.e., AWS Lambda), as they found that there is confusion regarding the broad definitions used to define term “serverless”. Table 3.4 summarizes their findings within these six themes.

Table 3.4. Summary of findings within the six themes Lietner et al. identified from their interviews [150].

Theme	Finding
Mental models	Developing serverless and FaaS applications requires a different conceptual model that emphasizes combining microservices which are either written by developers themselves or provided by the cloud infrastructure.
Types of serverless applications	Using FaaS is more suitable for event-driven applications rather than for applications that are long-running or have strict performance requirements. FaaS is frequently used with traditional cloud services (e.g., VMs, containers).
Application patterns	FaaS applications are small, frequently composed of 10 functions or less. Developers use application patterns to deal with the limitations of FaaS platforms.
Development languages and patterns	JavaScript, Python, Java, and C# are the most frequently used implementation languages for FaaS services.
Advantages and challenges	FaaS provides advantages related to business (i.e., reduced costs and increased developer productivity). The main development challenges of FaaS applications are vendor lock-in, integration testing, performance issues as well as the lack of sufficient tooling and documentation.
Deployment costs	Although FaaS services often reduce costs, this may be due to free tiers and low-utilization use cases. Additional services can easily increase the total deployment costs for an application.

They recommend that FaaS providers focus on better tools related to testing and debugging, such as tools for cloud event record-and-replay testing, published cloud images that allow users to accurately reproduce cloud behavior locally, and debuggers connected to executing functions. They point out that Azure provides many of these features through their Visual Studio integration [162] and encourage other providers to support such features as well. They recommend that FaaS providers enable the reuse of functions and create function catalogs where existing functions can be discovered. They point out that existing function marketplaces such as the Amazon Serverless Application Repository [18] have begun this effort and propose that more mature marketplaces will increase function reuse. They recommend that FaaS providers focus on higher-level abstractions such as a general-purpose or domain-specific language that

compiles into a format that is deployable on serverless platforms.

Interview study reveals that IaC developers are required to use multiple tools as well as the most challenging tasks are testing and debugging.

IaC is frequently used in containerized development to automatically configure infrastructure [14, 12]. As well, we interviewed AI/ML containerized app developers employing Kubernetes clusters for various purposes, including to develop and deploy Ansible [17] apps, which automates and optimizes IaC [14].

Therefore, understanding the particular challenges that IaC development creates is relevant to the results of our study.

Guerriero et al. conducted interviews to identify IaC development practices and challenges, particularly regarding the complex process of understanding and employing multiple programming languages and tools [133]. They note that software development is increasingly focused on providing faster deployment and continuous software updates, as well as constantly evolving with market needs and employing scalable technologies such as cloud computing. They point out that this is reflected by the prevalence of DevOps practices that focus on shortening the software development cycle.

They define IaC as a DevOps practice for automatically managing reusable scripts of infrastructure code, instead of being done manually (i.e., slower, more time-consuming, and error-prone) by system admins. They point out that many platforms have been developed to support different programming languages and help manage various parts of IaC development. They share several examples, such as tools for provisioning and orchestrating VMs (i.e., Cloudify [10], Terraform [69]), similar tools for container technologies (i.e., Docker Swarm [67], Kubernetes), and configuration management tools (i.e., Chef [27], Ansible, Puppet [59]). Although they note that such a range of IaC programming languages and tools is a result of interest, they emphasize that understanding and employing a multiplicity of technology is complex. They further emphasize that understanding the development practices and challenges faced, particularly regarding the tools used, can inform practitioners on what to focus on improving regarding

tooling.

They conducted interviews to understand developer perspectives on the best practices for IaC development, how tools support IaC development, and the challenges faced in IaC development. They reached out to senior developers from 86 companies and recruited 44 of them to conduct semi-structured interviews. The interviews were conducted by two authors remotely over Skype. Table 3.5 includes the general structure of the interviews conducted by Guerriero et al. [133]. The interviews were recorded and transcribed for analysis.

Table 3.5. General topics and questions from semi-structured interviews with senior IaC developers conducted by Guerriero et al. [133]. For questions Q12–Q17 they defined a Likert scale with five values ranging from “Not Important” to “Extremely Important”.

Topic	Questions
Background	1. How many years of working experience do you have in DevOps? 2. Do you actively develop IaC? 3. How many hours per week do you work on developing IaC? 4. Do you also develop standard (non-IaC) software?
IaC Development Practices	5. Describe the main differences between writing standard code and IaC. 6. Do you have anything you consider a bad practice in IaC development? Please specify. 7. Do you have any good practice to develop IaC? Please specify.
IaC Development Tooling	8. When you develop IaC, which specific software development tools do you use? (e.g. IDEs, static analyzers, debuggers, testing frameworks, documentation frameworks)? 9. Which DevOps tool(s) are you experienced with? 10. For each tool you are experienced with, could you name some pros and cons in terms of the programming language for writing IaC it provides?
IaC Development Challenges	11. Could you list from 2 to 5 common issues that you currently face when you develop IaC? 12. How desirable are IaC-specific Integrated Development Environments (IDEs)? 13. How desirable are IaC-specific Integrated testing frameworks? 14. How desirable is IaC language standardization? 15. How desirable are IaC-specific static analysis tools? 16. How desirable are tools and methods to support IaC security and privacy-related issues? 17. How desirable is to have IaC support for heterogeneous infrastructures (e.g., GPUs, FPGAs, Cloud, HPC)? 18. Could you mention any other challenge and future direction in the context of IaC development?

They analyzed the interview transcripts using a content analysis method [134]. Initially, four authors independently reviewed each interview transcript and assigned codes to relevant information. Then, all together they discussed their codes and finalized the relevant information to be discussed.

They presented results on the differences between IaC code and other, general source code. Regarding these differences, they found that IaC code is declarative in nature, compared to the imperative nature of general source code. As a result, they found that this changes the way in which developers understand and design IaC code. Other differences they found include disadvantages regarding testing and debugging. Regarding disadvantages in testing, they found a lack of standard practices and proper testing environments. Regarding disadvantages in debugging, they also found a lack of standard practices and that challenges are compounded by the distributed nature of IaC development. They also found that IaC development has longer feedback loops as developers must wait for the infrastructure to be deployed before they can evaluate whether their code is working correctly.

They presented results on the bad and best practices of IaC development. Regarding the bad practices, they found the most frequent to be hard-coding values in scripts such as credentials and constants. They conclude that hard-coding impacts maintainability (i.e., lack of environment variables) and can lead to security issues (i.e., embedding passwords, ssh keys, access codes). The next most frequent bad practice they found is that IaC code is too polyglot, with various languages being used for the same environment. They conclude that IaC code that is too polyglot impacts clarity and maintenance. Similarly, they found poor documentation to be another bad practice that contributes to the impact of polyglot IaC code.

Regarding the best practices, they found the most frequent is to make use of secret-injection to create IaC scripts such that the orchestrator or user can configure the desired secret, such as the environment variables, when needed. The next most frequent best practice they found is to develop and test in small increments and configure the deployment to be as deterministic as possible. They also found best practices such as reuse by abstraction and low nesting of IaC

code to make templates more modular.

The presented results on the various support tools used in IaC development and IaC-specific languages and tools. Regarding frequently used support tools for IaC development, they found the majority are IDEs, followed by text editors, linters, and models (i.e., UML diagrams, dependency graphs). Regarding frequently used IaC-specific languages and tools, they found that no IaC tool is used by more than 60% of participants, with Docker being the highest majority (59.5%). They conclude that this demonstrates a multiplicity of IaC technology that is complex, with many solutions and no single tool that dominates the market.

They found 8-10 tools that constitute equally-used alternatives in IaC development, including Docker, Ansible, Vagrant, Kubernetes, and more. They found that 69% of participants use three or more different tools. They conclude that this demonstrates the absence of a singular tool that integrates the various parts of IaC development. They further conclude that as a result, developers select multiple tools that have various purposes and combine them for IaC development. They found that the most frequently used tools that are employed by 30% of participants (i.e., Kubernetes for container orchestration, Vagrant for VM management, Chef and Ansible for service configuration management, Terraform for orchestrating services, Docker for building containers) deal with various parts of IaC development. They note that IaC code is polyglot by design, and recommend that practitioners create an ontology of IaC with representations and definitions of the categories, properties, and relationships among concepts. They conclude that such an ontology could help IaC developers understand how tools are used for various parts of IaC development and make informed decisions regarding tool combinations.

They categorized the main benefits and challenges found regarding support tools into six categories: 1) coding, 2) portability, 3) automation, 4) usability, 5) extensibility, and 6) maturity. Overall, they found that there is no universal tool for IaC development and developers must effectively combine multiple tools through experience. They found no consensus among participants regarding the benefits of these tools. They found the main challenges to be the testability, readability, consistency, and portability of these tools. They reported on the most

frequent challenges of IaC development. They found the most frequent challenge to be testability and the lack of a runtime environment in which IaC code can be tested.

Regarding what participants consider a priority to improve support for the IaC development cycle, they had participants answer questions on Likert scale, as explained in the description of Figure 3.5. They used a triangular membership function [174] to assign a numerical value in the normalized range [0,1]. They found that a majority of the responses indicated that testing frameworks are extremely important (0.90). They also found that participants perceive static analysis tools (0.82), languages standardization (0.80), security and privacy (0.76), support for heterogeneous infrastructures (0.75), as well as ad-hoc IDEs (0.74) to be highly important.

Regarding the future of IaC development, they found that serverless computing was mentioned with apprehension regarding the responsibility of developers to create and manage various application code functions. They also found that the increasing number of tools and changes in development processes and organizations are future concerns. They also note that despite programming language standardization efforts behind OASIS TOSCA [6], TOSCA-enabled orchestrators are not yet the standard solution for IaC development. Overall, they found that TOSCA is employed by 20% of the participants. They conclude that this identifies either a fault of the standard in terms of its dissemination and usage, or a divergence from the requirements in practice.

3.2.1.3 Takeaways on General Cloud Development

These prior works contribute to understanding several perspectives within general cloud development [104], including IaC development [133], configuration design and implementation [210], as well as serverless and FaaS development [207, 150]. In general, they found that general cloud development results in a DevOps style of communication that leads to frequent collaboration between developers and operations engineers with varying job role expertise [104]. Furthermore, they found that monitoring tools are built to be used by the operations engineers, rather than the developers who, without operations expertise, find these existing metrics difficult

to leverage.

Regarding general cloud development and IaC development, they both found that developers must use multiple tools that have various purposes (i.e., performance tasks, log management, analytics) and combine them through experience, as there is no singular tool that integrates the various development tasks [104, 133]. They also found little overlap in the tools used by IaC developers, which suggests the existence of many alternatives and a lack of standard practices [133]. Regarding IaC development, they found the most challenges tasks include debugging and testing due to a lack of standard development practices, compounded by the distributed nature of IaC development as well as longer feedback loops as developers must wait for the infrastructure to be deployed before they can evaluate whether their code is working correctly [133].

Regarding configuration design and implementation, they found that there is a need for standard practices and automated configuration tools to choose and test parameterization and default values to help developers make decisions about which constant values need to be parameterized, understand when default values will lead to system failures, as well as more thoroughly test and benchmark default values [210]. Regarding serverless development, they also found that there is a need for tools and standard practices to help developers debug configuration errors [207]. Regarding serverless and FaaS development, they found that despite the abstraction of most operational tasks, development is more complex [207, 150]. In support of this, they identified many challenges that serverless developers face, including the frequently performed task of integrating multiple services that involves developers designing functions to combine services that they either wrote themselves or were provided by the cloud infrastructure [207, 150].

3.2.2 AI/ML Cloud Development

Although, to our knowledge, there are few user studies on AI/ML cloud development, there are other interesting research studies related to this subject that provide relevant context for understanding the results of our study, such as large-scale analyses. Below, we discuss one large-scale analysis [109] and user study [194].

3.2.2.1 Large-Scale Analyses

Large-scale analysis of computer vision (CV) cloud services-related Stack Overflow posts finds that developers lack AI/ML cloud development experience and require more supportive documentation.

Understanding the particular challenges that AI/ML service API development creates is relevant to the results of our study, as we focus on AI/ML containerized app developers employing Kubernetes clusters for various purposes, including to develop and deploy CV applications.

Cummaudo et al. conducted a large-scale analysis of SoF posts to understand the challenges application developers face implementing web APIs for AI/ML cloud services, including how they may not be aware of their limitations or how they work given that they are not AI/ML experts and lack the necessary background knowledge, as such details are not fully documented [109]. They note that APIs for AI/ML cloud services abstract complex techniques that are data-dependent, probabilistic, and potentially nondeterministic, replacing them with simpler API calls such that the AI/ML cloud development experience necessary for creating and implementing these services is abstracted away. For example, they note that these services abstract away information regarding the training algorithms and datasets, evaluation procedures, as well as optimizations. Furthermore, they note that these services frequently do not offer a properly versioned end-point such that cloud providers may change service behavior without clarity.

They point out that although the documentation for cloud services consists of a brief explanation of the RESTful end-points along with the expected inputs and responses, it does not explain these complex techniques. They note that these services are marketed to application developers that don't fully understand how the API components are created (i.e., assumptions in training data and algorithms), how they can fail, and how to appropriately use them. They emphasize that application developers without experience in the complex techniques used by these AI/ML cloud services may compromise the software quality of the applications they develop that depend on these techniques. They emphasize that understanding what developers

discuss on platforms such as SoF can lead to better insights into the challenges they face and improve the quality of APIs for AI/ML cloud services. They further emphasize that such an understanding can inform API designers on what to improve regarding documentation, education, and the design of end-points.

They conducted a large-scale analysis of SoF posts to understand the challenges faced by developers implementing AI/ML cloud services. To create the dataset, they captured SoF posts related to CV services by identifying issues related to four prominent AI/ML cloud service providers, including Google Cloud [31], AWS [16], Azure [22] and IBM Cloud [46]. They note that they focused on CV services because of their maturity, stability, and popularity.

To capture CV-related posts, they first defined a list of services to use as search terms on the Stack Exchange Data Explorer (SEDE) [64]. To compose this list, they initially conducted extensive Google searches, conducted in January 2019, and manually reviewed relevant cloud service provider websites, resulting in a list of 91 services. Then, they manually searched each term on SoF to determine its relevance and compile a different list of user-defined tags, resulting in a list of 31 tags. Using these tags, they manually searched the SEDE, filtered out unanswered posts, and extracting a random sample of 100 questions, 400 for each service, to manually review. Their final list of services consisted of 229 terms relevant to services provided by those four cloud providers.

In May 2019, they used this list to search within the body or title of SoF posts via Google's BigQuery platform [24], which resulted in an initial dataset of 21,226 posts. To filter and clean the extracted posts, they removed abbreviations from their list of search terms and consolidated overlapping terms. As they wanted to conduct manual qualitative analysis, they reduced their list of search terms to 27, which only included CV services. Their final dataset included 1,425 posts, extracted in June 2019.

To categorize the posts, they applied two existing taxonomies used in prior work: 1) documentation-specific [72] and 2) generalized [89]. They used a documentation-specific taxonomy to categorize issues that result from documentation related to several themes, including

information content (i.e., what is written and how it is organized), process-related, and tool-related themes. They used a generalized taxonomy to cover a broader range of issues related to six themes: 1) API usage and changes, 2) discrepancy (i.e., the expected compared to the actual behavior of the API), 3) errors, 4) review, 5) conceptual, as well as 6) learning. Four authors manually categorized 1,375 posts and the remaining 50 posts were categorized nine times as part of a reliability analysis, resulting in 1,825 categorizations made. All coders met to agree on a shared interpretation of the original definitions of the taxonomy themes, which were then paraphrased. They further conducted statistical calculations of reliability in which each category was assigned a value.

Given that multiple authors coded the data, they conducted intra- and inter-rater reliability to measure consistency and applied Light's Kappa [152] to their ratings to produce an overall measure of agreement. They were guided by the prior work of Landis and Koch [148] in interpreting the kappa reliability statistics, where $0.00 \leq \kappa \leq 0.20$ indicates slight agreement and $0.21 \leq \kappa \leq 0.40$ indicates fair agreement. They found that although all coders agreed on a shared interpretation of the taxonomies, inter-rater agreement measures aligned slightly (0.148) for the documentation-specific taxonomy and fairly (0.295) for the generalized taxonomy.

They found that in general, developers do not ask questions exclusively related to documentation, as the documentation-specific taxonomy was only able to classify 10.30% of the posts. Rather, they found that questions about documentation are frequently combined with code and other context. Regarding the posts categorized as documentation-specific, they found that a majority (47.87%) ask for additional information to understand the API. They conclude that this indicates a lack of completeness in the documentation, as it does not adequately cover domain-specific issues that influence the software quality of AI/ML cloud service APIs (i.e., exploration, data engineering, distributed infrastructure, training data, model characteristics). They also found posts that revealed issues with documentation accuracy (22.87%) and confusion regarding documentation (23.94%). They conclude that the posts regarding confusion indicate issues of documentation usability (e.g., readability, usefulness).

They found that the generalized taxonomy was able to classify 86.52% of posts. Regarding the posts categorized using the generalized taxonomy, they found that a majority (32.05%) are error-related, with API-usage questions being the second highest (22.29%) category. They also found discrepancy- (16.34%), review- (15.14%), and conceptual-related (11.02%) questions. They conclude that the API-usage, as well as the discrepancy- and conceptual-related questions indicate an insufficient understanding of the behavior, functional capability, and limitation of these services. They conclude that the review-related questions provide support in understanding the context to which these services work.

They further analyzed the reason for the issues expressed by the developers in these posts using two learning taxonomies: Bloom's context complexity and intellectual ability taxonomy [92] and the SOLO taxonomy [90]. Regarding Bloom's taxonomy, they note that these posts are related to six levels of learning: knowledge, comprehension, application, analysis, evaluation, and synthesis. Regarding the SOLO taxonomy, they note that these questions are related to five levels of understanding: pre-structural, uni-structural, multi-structural, relational, and extended abstract. They categorized those remaining 50 posts using these two taxonomies. They found that a majority fall into the lower-levels of Bloom's taxonomy, including knowledge (35.79%) and comprehension(29.47%) as well as the lower-levels of the SOLO taxonomy, including pre-structural (57.37%) and uni-structural (30.79%).

They conclude that developers are struggling due to a lack of background knowledge (e.g., vocabulary). They conclude that these lower-level issues are solvable through improving documentation (e.g., better descriptions of response data and error messages). They recommend that such API providers include overview documentation and not only endpoint documentation. Furthermore, they recommend that documentation should be written such that developers without experience in AI/ML cloud development can understand and apply it in the context of their application. They recommend an interactive tutorial to help these developers understand and explore the capabilities and limitations of these APIs in a hands-on approach. However, they conclude that the higher-level issues, including the application (30.53%) and analysis (4.21%)

levels from Bloom’s taxonomy, as well as the multi-structural (5.26%) and relational (6.58%) levels from the SOLO taxonomy are not as easily solvable and would require improved education practices. To this extent, they recommend the development of a software engineer’s “crash-course” in AI/ML.

3.2.2.2 User Studies

Mixed-methods user study reveals how environmental scientists developing and deploying models perform tasks that diverge from their expertise.

Understanding the practices of scientists from a broad range of environmental science fields that develop or use environmental models for various purposes (i.e., climate, insect populations, weather, and soil chemistry) is related to our study, as we focus on AI/ML containerized app developers employing Kubernetes clusters for purposes related to their job role expertise, including data scientists developing and deploying environmental model applications.

Simm et al. conducted interviews, participant observations, and a case study to understand the challenges faced by environmental scientists in developing and deploying models, including the complexity that comes from: 1) working with large-scale datasets, 2) efficiently using computing resources, 3) deploying to various computing architectures, and 4) combining different models [194]. For example, they note that models frequently become coupled to specific architectures as they evolve with the scientist’s understanding of both the model and the underlying software and architecture. They note that in general, although basic environmental models may be run locally, more complex models require a dedicated HPC system. Regarding more complex models, they note that different model simulations are frequently combined and require significant computing resources, particularly for climate prediction. For example, they note that such models may be run multiple times with various starting points and assumptions, or multiple models may be run that were created at different organizations using their own interpretations and assumptions to predict the same output variables. They note that combining different models mitigates uncertainty (i.e., chaotic nature of environmental systems and lack of data for initial conditions), which is one of the main challenges in developing environmental models. Regarding

model deployment, they emphasize that it is necessary to increase the abstraction of underlying cloud services for managing distributed computing, which will allow scientists to focus on their job role expertise rather than specifics of the underlying computing infrastructure.

They conducted interviews, participant-observation, and a case study to understand the development practices and associated system admin skills and find opportunities for applying software engineering practices in this domain. They conducted semi-structured interviews with five environmental scientists from academic and industry-oriented research organizations. Participants were from a broad range of environmental science fields and developed or used environmental models for various purposes (i.e., climate, insect populations, weather, and soil chemistry). During the interviews, participants were asked to introduce and demonstrate the model they work with or on, show the code and scripts that they had developed themselves, as well as run the model. Moreover, participants were prompted with a question framework if certain topics were not covered naturally in conversation, shown in Table 3.6. The interviews lasted 1.5-3 hours in total. They interviewed participants in the familiar context of their work place. Interviews were audio and video-recorded to capture the conversation and the participant's computer screen. The interview data was coded using a Grounded Theory approach.

Regarding development practices, they found that version control supports collaboration and helps with understanding any code changes. They found that models are frequently written in scientific as well as mathematical languages (i.e., Fortran, Matlab, R) and Python is used to analyze output data. They found that the models were written in these scientific and mathematical languages such that the programs are monolithic, without defined interfaces, and have little consideration for code or cloud service reuse. As a result, they found that this restricts the ability for each component of a model to be deployed appropriately.

Regarding deploying models on various architectures, they found that participants rarely considered or understood how to accomplish this. They observed that the participants wrote models on a local machine, then adapted them at some level to run on HPC systems. Despite how frequently models are combined, in general, they found that models are not developed with this

Table 3.6. Semi-structured interview question framework used to prompt participants for the interviews conducted by Simm et al. [194].

Question Topic	Description of Prompt
Individual	• Types of model they work with • How frequently they work with models • Programming languages, tools, and frameworks used to write or adapt models • SE background and experience
Model Specifics	• How the model runs and is deployed • Architecture and computational requirements of the model • Whether the model is designed as a stand-alone module or combined with other models
Automation	• Availability and description of workflows, scripts, or GUIs to run the model
Architecture	• Computing architecture the model runs on (i.e., distributed, shared, and cluster)
Data Formats and Storage	• Type and formats of input and output data required to execute model simulation • Hardware and software requirements for storing the input and output data
Preprocessing and Deployment	• Requirements or pre-requisite tasks related to data, model or model deployment
Processing	• Duration of different model simulations • General practices to keep track of running simulations
Analysis and Visualization	• Tools used to analyze and visualize model outputs from the data produced
Limitations and Enhancements	• General practices followed • Challenges faced • Various aspects of human values

in mind. Regarding the participants' associated system admin skills, they found that a high-level understanding is required when installing, configuring, and deploying models. They also found that each of the participants was self-taught regarding system admin and programming skills, as well as none had a formal education in computer science or software engineering.

They found opportunities to apply software engineering practices in all levels of model development. Regarding abstraction, they conclude that Model Driven Engineering (MDE) provides the tools through Domain Specific Languages (DSLs) to be able to increase the level of the abstraction of underlying cloud services such that scientists do not need to become system admin experts. Regarding framework support, they conclude that frameworks and environments should be developed to support the separation of scientific tasks from data preparation. They emphasize that this would allow each component of a model to be developed in an appropriate language and deployed on the most appropriate architecture. They conclude that developing models with defined interfaces would allow them to be combined with other models. They also conclude that frameworks and environments should support the scaling of models from local machines to cloud and HPC systems. Regarding flexible architectures, they conclude

that such framework and abstraction support using DSLs may facilitate deployment to various architectures (i.e., on demand computing in the cloud, GPU nodes).

They also presented a case study to demonstrate how DSLs can be used to abstract away the time-intensive tasks related to managing distributed computing that diverge from environmental scientists' expertise by automating the deployment of the weather research and forecasting (WRF) model [195]. The WRF model is used for atmospheric research and forecasting, as well as provides simulations for a large variety of applications (i.e., regional climate, air quality, urban heat islands, hurricanes, forest fires, flooding through combined hydrological models). They note that models such as the WRF are traditionally run locally or on HPC systems. They note that the WRF model was chosen for the case study for four reasons, including that: 1) its open-source, 2) has a large community of users, 3) its installation is challenging, as well as 4) deploying the model using cloud resources would enable users to conduct new simulations, taking advantage of on demand scalability, accessibility, and cloud services such as data stores and analytics. Although the WRF is a numerical weather prediction (NWP) and atmospheric simulation system that does not use ML, we include this study here as the WRF model is frequently combined with other AI/ML models that increase the accuracy of its forecasts [158, 190, 110].

They evaluated the deployment's cost and performance by running the same simulation (i.e., a 24-hour WRF simulation centered over the Lagos, Nigeria metropolitan area using the WRF version 3.9.1.1) on a HPC cluster and a Microsoft Azure cluster using varying numbers of nodes (i.e., eight-, 16-, 32-, 64-bit processors). No GPU nodes were used. They note that comparable performance is not expected as the Azure cluster is composed of general purpose nodes rather than dedicated HPC-supported nodes (i.e., high InfiniBand [1] access). Indeed, they found that the performance of the HPC cluster was always better than the Azure cluster on standard nodes. They also found that there is a continuous reduction in execution time for both HPC and Azure as the number of processors increases from 8 to 64. They conclude that a cluster composed of general purpose nodes can be used to effectively run the WRF simulation.

They calculated the cost of running the same simulation on varying numbers of processors on the Azure cluster. They found that the minimum cost is less than \$1 and the maximum is about \$3. They conclude that scientists without access to HPC clusters, or who do not want to wait in a queue to use such clusters, can affordably run their models using such a cluster. They note that cloud deployment is on demand, with no queue time required, such that experiments can be instantly run. Regarding cost and performance, they found that the execution time of running the simulation on 32- and 64-bit processors is comparable, however 64-bit processors have a higher cost. They conclude that there is an optimal threshold of assigning the number of processors to a WRF simulation, depending on simulation size.

Regarding other qualities of the WRF model deployment beyond cost and performance, they conclude that the complexity of the deployment was reduced by abstracting away the WRF model installation and deployment tasks. They abstracted away these tasks using automated scripts to manage all dependencies such as downloading libraries and setting up the environment for running a WRF simulation. The scripts can also deploy the WRF model on a private Linux machine or a VM on a public cloud provider's infrastructure. They created a walk-through guide to enable users without cloud computing expertise to create a Microsoft Azure's cluster for the WRF deployment. Users are also provided with a guide for creating machine images for reproducibility.

They collected feedback on the deployment from six expert WRF users in a group discussion regarding its potential impacts and benefits to their work, as well as applicable use cases. These expert users all frequently run research experiments using the WRF model. Regarding applicable use cases for the automated deployment, they discussed three types of users: 1) novice users (i.e., Masters students that spent several months learning how to install and configure the model conducting any research), 2) users that want to run the model in using a standard process and in combination with other models, as well as 3) expert users who want a quicker cloud deployment option that doesn't require waiting in a HPC cluster queue and could benefit from parallel execution. They discussed how a DSL may need to cater to these types of

users, including how expert users may benefit from a different kind of DSL for architecting cloud deployments, or simply guidance on cloud deployment as they are more familiar with CLIs and scripting.

Similar to how they address varying levels of AI/ML cloud development experience in the solutions demonstrated and proposed in this user study, our study seeks to understand the specific AI/ML containerized app development tasks and concepts that developers with varying expertise find challenging. These challenges lead to educational materials, presentations, or more comprehensive documentation to be created, as discussed in Chapter 5. In general, a deeper understanding of how scientists with varying levels of AI/ML cloud development experience use such an automated deployment process in practice could help to further evaluate its qualities and improve it.

Along with conducting their own broad user study, they cited an in-depth user study conducted by Easterbrook and Johns [117] that conducted participant observation over eight weeks at a national weather forecasting agency to understand their software engineering practices and the models being developed and used. Easterbrook and Johns found that environmental model coding is frequently done by scientists without a software engineering background. Easterbrook and Johns found that the general structure for large “community models” consists of a small number of code owners and a larger number of contributors. Easterbrook and Johns note in an example that in general, community models are split into components that represent physical boundaries and different research communities (i.e. ocean, atmospheric science). Easterbrook and Johns found that developing community models can take decades and involve hundreds of scientists. Easterbrook and Johns found that community models can consist of millions of lines of code and could benefit from software engineering design practices.

Abstracting and automating the tasks related to managing distributed computing that diverge from scientists’ expertise is particularly salient to our study’s results. We find that our participants use similar methods, such as scripts, to achieve the abstraction of these time-intensive tasks. The specific AI/ML containerized app development tasks that are abstracted and automated

by developers, their solutions, as well as their implications are discussed in Chapter 5.

3.2.2.3 Takeaways on AI/ML Cloud Development

These prior works contribute to understanding several perspectives within AI/ML cloud development, including environmental model development [194] and cloud service API development [109]. Regarding environmental model development, they found that a high-level understanding of system admin job role expertise is required when installing, configuring, and deploying environmental models [194]. Furthermore, they found that environmental scientists have no formal system admin or software engineering training and are self-taught in these skills. To address this knowledge gap, they advocate for increasing the level of abstraction in underlying cloud services such that environmental scientists do not need to become system admin experts. However, regarding AI/ML cloud service API development, they found that the developers using such abstracted cloud services which hide the complexity of the techniques for training algorithms and datasets, evaluation procedures, and optimizations, don't fully understand how the API components are created, how they can fail, as well as how to appropriately use them [109]. Therefore, they found that level of abstraction offered in these cloud services should consider users with varying levels of AI/ML cloud development experience (i.e., novice, intermediate, expert) [109].

3.2.3 Takeaways on Cloud Development

Many of these prior works emphasize the challenges that cloud developers face in understanding fundamental concepts [207] or adhering to new mental models [150] and point out existing problems with technical documentation [207, 150, 133, 109] such as a lack of accuracy and effectiveness, as well as how this is a long-standing problem in software engineering [210]. As well, these prior works highlight new challenges in the software development cycle that have been introduced by cloud development, including organizational changes as well as changes in development processes (i.e., debugging and maintenance [104]) that result in a lack of standard

practices [133, 210], an increase in the number of tools used [104, 133], and the need for automation [210]. Furthermore, these prior works emphasize how these new challenges are compounded by AI/ML cloud development, despite an increase of abstraction [194, 109] and automation [194]. For example, changes in development processes (i.e., separation of scientific tasks from data preparation as well as the scaling of models from local machines to cloud and HPC systems [194]).

To the extent that these results generalize to our study’s domain of AI/ML containerized app development, we’d expect to find similar challenges. We could even expect to see that these challenges would be compounded, as the developers we interviewed all employ Kubernetes, which requires such system admin job role expertise, rather than employ an abstracted cloud service that purports to not require such expertise.

3.3 Developer Perspectives on Containerized Development

3.3.1 General Containerized Development

Although, to our knowledge, there are few user studies on general containerized development, there are other interesting research studies related to this subject that provide relevant context for understanding the results of our study. These broadly fall into three other categories that include large-scale analyses, surveys, and tool interventions. Below, we discuss one large-scale analysis [135], survey [120] and user study [211].

3.3.1.1 Large-Scale Analyses

Large-scale analysis of Docker-related Stack Overflow (SoF) posts finds that there are few Docker experts, as they require knowledge in multiple domains.

Docker is one of the main tools used in containerized development. Therefore, understanding the particular challenges that Docker creates is relevant to the results of our study.

Haque et al. conducted a large-scale analysis of SoF posts to understand the challenges faced by developers employing container technologies, such as Docker, including how con-

tainerized development requires knowledge in multiple domains such as networking, operating systems, cloud computing, and software engineering [135]. They point out that Docker has been increasingly used among developers due to several facts, including that Docker supports a suitable process for creating and building containers, promotes close cooperation between developer and operations teams, and enables CD processes. They emphasize that Docker is a fast-growing technology and as such, is important to identify and address the existing challenges that developers face in employing Docker. They further emphasize that without establishing this understanding, it is unclear where practitioners, researchers, and educators should focus their efforts to provide contributions that benefit the Docker community.

They conducted a large-scale analysis of SoF posts to understand what Docker topics developers ask questions about, the characteristics in terms of popularity and difficulty of the identified topics, and whether there are enough experts to address Docker-related questions. They point out that SoF provides a prolific source of data on the perspectives of developers, without organizational context. To create the dataset, they initially used a publicly available SoF dataset from SOTorrent [87]. From this dataset, they note that Google’s BigQuery platform has been used to extract 46,947,633 questions and answers posted over a time span of over 11 years, from August 2008 to December 2019 by 4,533,602 developers. Among these posts, 39.61% are questions and 60.16% are answers of which 20.72% are marked as accepted answers with an acceptance rate of 34.45%. They then identified a sub-set of SoF questions that discuss Docker-related questions using two filtering techniques: 1) tag- and 2) content-filtering. The final dataset consisted of 113,922 SoF questions and answers of which 71.4% are questions and 28.6% are accepted answers.

To organize the topics, they applied the MALLET [50] implementation of Latent Dirichlet Allocation (LDA) [91], a topic modeling technique used in prior work [86, 75]. They note that in general, MALLET generates a list of 10-word groups as topics and cannot infer meaningful topic labels. To meaningfully label topics, they used the open card sort method [121] and note that they were guided by prior work in implementing this method [86, 208]. There is no predefined

set of topic names in open card sort method. The open card sort method is similar to the open coding method used in our study and we were also guided by prior work in implementing our coding methods, as discussed in Chapter 4.

Following the open card sort method and guided by prior work [73], they inspected the top 10 words from each topic and read through the 15 most relevant posts for that topic to identify a topic name that best explained the words used. They also verified whether the chosen topic names were suitable by examining 15 randomly selected posts from each topic. They also used the open card sort method to organize similar topics into high-level categories. They organized the topics into 13 categories (i.e., application development, networking, configuration, basic concepts, database systems, debugging, resource management, docker desktop, repository, integration, orchestration, deployment, user authentication).

They found that more than half of the posts (56.9%) fall into four categories: 1) application development, 2) configuration, 3) networking, and 4) basic concepts. Table 3.7 summarizes these categories and the topics that they further identified within them. They note that in general, these topics and categories involve a combination of knowledge from different domains. They conclude that Docker development requires a developer to have knowledge in these multiple domains.

They identified the most popular topics based on: 1) the average number of views, 2) favorite count, and 3) score of a question. They found that the most popular topics include monitoring status, transferring data, and authenticating users. They identified the most challenging topics based on the: 1) percentage of questions of a topic that has no accepted answers and 2) average median time needed for questions of a topic to receive accepted answers. They found that the most challenging topics include web browsers, networking errors, and memory management. They point out that these metrics for identifying popular and challenging topics are well-known and used in prior work [86, 75, 208, 184].

They evaluated whether a lack of experts is one of the factors behind the difficulty in answering Docker questions. They define experts as the users who provide significant

Table 3.7. Further breakdown of the topics identified by Haque et al. within the categories of Docker-related SoF posts in the dataset collected by them for their large-scale analysis [135]. The category of basic concepts is omitted as it did not include any further identified topics.

Categories	Topics
Application development	• Framework management and development • Coding issues • Compilation • Data transfer • Web browser • Python
Configuration	• Logging • Web server and framework configuration • Commanding and scripting
Networking	• Stream processing • Container linking • Networking error • Connection management • Docker network

contributions as well as effective and helpful answers to SoF. To identify experts, they followed an approach used in prior work [201], in which user answers (i.e., quality of the answered question) and profile performance (i.e., user reputation and user profile view count) are derived from the SoF dataset and combined to form an expertise value for each user. They found that there is a substantial lack of Docker experts in the SoF community when compared to other areas, such as web development.

They further analyzed the most popular and challenging Docker topics as a tradeoff. They recommend that developers focus on topics that match their current skill levels. For example, they note that novices may benefit from learning about topics that have been well-discussed on SoF and issues that have already been solved. Likewise, they note that experienced developers may want to become more knowledgeable from learning about challenging topics that are also popular on SoF. They recommend that managers use such tradeoffs to assign tasks that match their teams' skill level. They conclude that practitioners may benefit from understanding the challenges surfaced by the SoF community for improving frameworks and tools that facilitate development when using and configuring Docker-related technology. Likewise, they conclude that academic

researchers may benefit from investigating problematic topics to deeply understand the challenges faced by developers. Furthermore, they conclude that educators may use such tradeoffs to select appropriate topics and time-frames for exercises and assignments in Docker-related subjects.

3.3.1.2 Surveys

SIG Apps surveys emphasize the need for custom tooling, data visualizations, and documentation for Kubernetes application developers and operators, cluster operators, as well as tool developers.

Kubernetes special interest groups (SIGs) such as the Apps Special Interest Group (SIG Apps) conduct surveys to gain insight into the Kubernetes developer and operator community regarding the tools, best practices, and UX of building and operating cloud native applications. In 2018, they conducted their most recent survey to understand how application developers and operators, cluster operators, as well as tool developers use or want to use Kubernetes [120]. The survey focused on these four user roles and the features and sub-projects owned by the Kubernetes organization (i.e, kubectl, Dashboard, Minikube, Helm, the Workloads API). They do not define these four user roles beyond asking respondents to choose between several options to describe their experience with Kubernetes, shown in Table 3.8.

Table 3.8. Answer choices for 2018 SIG Apps survey question, “I would describe my experience with Kubernetes as: ”, not including “None” or open-ended responses [120]

Answer Choices for 2018 SIG Apps Survey Question, “I would describe my experience with Kubernetes as: ”
• I operate Kubernetes clusters for my organization. • I operate Kubernetes clusters to support my application development. • I operate applications on Kubernetes. • I develop applications that interact with the Kubernetes API. • I develop extensions to Kubernetes. • I develop internal tools for running, managing, and validating applications that run on Kubernetes.

They found that the 2018 survey had 380 respondents from the Kubernetes user community, which is a substantial increase (679%) in the number of respondents from the last survey conducted in 2016, which had 56 respondents [119]. They found that a majority of respondents (72.6%) identify as both cluster operators and application operators. Furthermore, they found that even more respondents (77.2%) identify as both application developers and application operators.

Although this survey distinguishes between application developer, application operator, and cluster operator job roles in a more general context, this separation is not reflected in the job roles reported on in our interviews with those employing Kubernetes in the specific context of AI/ML containerized app development. This is partly due to how we identify the tasks that correspond to participants' job roles, as defined in our initial survey, presented in prior work by Arroyo et al. [80], which we later discuss in Chapter 3.3.2.1. This could also be because our participants, despite being within the same community of Kubernetes users, likely come from different organizations that have their own defined job roles. We define the job roles reported on in our interviews in Chapter 4.3 and discuss our findings regarding these job roles in Chapters 5 and 6.

Regarding custom tooling, they found that two thirds of respondents invent their own custom tools to help with application development and operation. From the 44% of respondents that shared their reasons for inventing custom tools, they found that custom tooling helps to simplify workflows for multi-cluster application deployment and provide integration with existing tools. Respondents also shared that they invent custom tools because they were unable to find an existing tool to fill their needs. Regarding application management, they found little overlap in the specific application management tools used by developers. For example, they found that only four tools were in use by more than 10% of respondents, one of them being Helm, which they found to be in use by 64% of those respondents. In total, they found over 80 tools in use.

Regarding data visualizations, they found that most respondents would appreciate the ability to visualize all of the Kubernetes objects that make up their application (i.e., 60% reported usually and 35% reported sometimes) and the objects that make up a sub-component of their application (i.e., 42% reported usually and 49% reported sometimes). Regarding documentation, they found that most respondents would benefit from documentation to better understand how to employ Kubernetes for running their applications in practice (i.e., 56% reported usually and 33% reported sometimes), and would benefit from improved clients, SDKs, or documentation that explained how to build tools that integrate with Kubernetes (i.e., 53% reported usually and

31% reported sometimes). Similarly, in the 2016 survey respondents shared that there is a lack of useful documentation [119].

The SIG Apps surveys do not ask about the development task context in which tools are used aside from application management and continuous integration and delivery (CI/CD). In our study, our findings confirm the challenges reported in these prior surveys and provide additional details about the nature of these challenges, as discussed in Chapter 5. For example, these surveys do not address how the reported challenges are compounded by AI/ML containerized app development. This is still a relatively new use for Kubernetes, as stated in a 2025 interview with the chairs and technology leads of SIG Apps, where they discuss AI as an upcoming trend in application management within Kubernetes [172].

3.3.1.3 User Studies

Mixed-methods user study reveals how DevOps and microservices impact collaboration among different teams.

DevOps is a main component of cloud development and IaC. As well, containerized development employs a microservices architecture. Therefore, understanding the particular challenges that DevOps and microservices create is relevant to the results of our study.

Zhou et al. conducted interviews and participant-observation to understand the challenges faced by developers employing DevOps and microservices, including the technical and organizational requirements of implementing them together [211]. They note that DevOps and microservices address current software demands, enable efficient and reliable development, as well as are generally employed together. The objective of this ethnographic study is to provide an in-depth perspective by participating in development as a member of the team to observe DevOps and microservices practices. They emphasize that focusing on a team's daily work may offer an insight into the organizational structure, advantages, and challenges of employing DevOps and microservices together, in practice. They further emphasize that organizational structure has become a challenge to employing DevOps and microservices in organizations where support for technology-driven changes and cultural shifts are required. They found that as

a result, developers from different teams (i.e., product, UI design, development, test, as well as operation and sale) frequently meet virtually to solve the problems encountered.

They conducted interviews and participant-observation to understand how organizations arrange software teams for DevOps and microservices, the impact of DevOps and microservices on teamwork, and the advantages and challenges of employing DevOps and microservices [211]. They conducted interviews and participant-observation in the three organizations of various sizes, including more than 1000 developers from the first organization, 100-1000 developers from the second organization, and 50-100 developers from the third organization. They conducted semi-structured interviews with developers that had various job roles, including one developer and four software architects from the first organization, one developer and two managers from the second organization, as well as one developer/software architect, one product owner, and one CTO from the third organization. They conducted both individual and group interviews with these developers. In total, they interviewed 11 developers. They conducted the participant-observation in three time periods over seven months, from January-December 2020.

They found that the main advantages of employing DevOps and microservices include rapid iteration and deployment, enhanced ability and skills, as well as reduced burden. Regarding enhanced ability and skills, they conclude that this results from requirements that demand a team be responsible for tasks across the entire development cycle of DevOps. Moreover, they conclude that this may result from the frequency in which DevOps developers change teams, and in turn perform different development tasks, learning new skills. Regarding reduced burden, they conclude that microservices allow developers to focus on specific domains.

They found that the main challenges of employing DevOps and microservices include the high cost and a lack of practical guidance. Regarding the high cost, they conclude that this results from the consumption of more resources by microservices. Moreover, in the first organization they found that the cost of deployment is constantly increasing, multiplied by the number of environments, as a result of a lack of deployment design. They further conclude that the high cost results from the time it takes to locate issues in microservices, as data consistency

must be guaranteed. Regarding a lack of practical guidance, they found that existing approaches for microservices decomposition, including Domain-Driven Design (DDD) [177] and dataflow-driven design [151], are limited by organizational changes that frustrate developers and require them to perform complex merging or splitting of domains.

They further conclude that these challenges are a result of two practices: fragmentary DevOps and the abuse of microservices. Regarding fragmentary DevOps, they identify two gaps in the DevOps development cycle. The first gap occurs between planning and coding tasks. In organization two they found that there is a lack of correlation between the requirements and the code. Similarly, in organization three they found that there is a lack of correlation between the bespoke project management system and the code. The second gap occurs between operations and deployment tasks. In organizations two and three, they found that infrastructure operations are independent from their development.

They also found that although all three organizations have knowledge-sharing platforms (i.e., presentations), teams prefer to share tools and methods within their own team rather than across teams. They conclude that this may intensify the learning curve experienced by different teams. Regarding the abuse of microservices, they conclude that the *"blind pursuit of fancy technologies"* results in counterproductive decisions for implementing microservices.

They made recommendations for practitioners employing DevOps and microservices. To address challenges from organizational structure, they recommend that technology-driven changes and cultural shifts be supported at the organization level (i.e., managers) rather than the individual team level. To address challenges from fragmentary DevOps, they recommend that teams establish the connections among the requirements, design, and code as well as consider operations in development. To address challenges from the abuse of microservices, they recommend that practitioners make architectural decisions that match their domain.

3.3.1.4 Takeaways on General Containerized Development

These prior works contribute to understanding several perspectives within general containerized development, including Docker development [135], Kubernetes development [120], as well as DevOps and microservices development [211]. Regarding Docker development, they found that there are few Docker experts as they require knowledge in multiple domains (i.e., networking, operating systems, cloud computing, and software engineering) [135]. Regarding Kubernetes development, they found that developers perform multiple job roles that require varying domain knowledge expertise, including performing both cluster operator and application developer job roles [120]. They also found that Kubernetes developers must use a multiplicity of tools, including their own custom tools, for application development and operation. Furthermore, they found little overlap among these tools, which suggests the existence of many alternatives and a lack of standard practices.

Regarding DevOps and microservices development, they found that the main advantages of employing DevOps and microservices include rapid iteration and deployment, as well as enhanced developer ability and skills [211]. Regarding rapid iteration and deployment, although this advantage is applicable to general containerized development, AI/ML containerized apps have longer feedback loops as developers must wait for resource- and time-intensive workloads to finish running before they can evaluate whether their code is working correctly, as discussed in Chapter 5. Regarding enhanced developer ability and skills, they found that developers must be responsible for tasks across the entire development cycle of DevOps and that they frequently change teams, and in turn perform different development tasks, learning new skills [211]. However, new skills come with their own learning curves and they found that in practice, the DevOps development cycle fragments operations and deployment tasks such that infrastructure operations are independent from their development [211]. As a result, they found that such an organizational structure leads to frequent meetings that require many experts from different teams to join in order to solve the problems encountered.

3.3.2 AI/ML Containerized App Development

Although, to our knowledge, there are no user studies on AI/ML containerized app development, there are interesting research studies related to this subject that provide the most relevant context for understanding the results of our study. These broadly fall into two categories that include surveys and tool interventions. Below, we discuss two surveys [33, 80] and seven tool interventions [149, 157, 125, 127, 129, 156, 164].

3.3.2.1 Surveys

CNCF surveys emphasize organizational challenges in containerized development and early AI/ML adoption.

The CNCF conducts an annual survey to gain insight into the cloud native community regarding adoption, cloud and container usage, Kubernetes reach, and project growth [33, 13, 8, 4]. In 2025, they conducted their latest survey with 628 respondents [33]. The respondents job roles included software developer/engineer (45%), systems admin (18%), professional services/-consultant (13%), manager/director/VP (7%), scientist/researcher/specialist (3%), executive (3%), network admin (1%), teacher (1%), and other (8%). A majority of the respondents (60%) reported having more than 10 years of professional IT experience.

In the 2025 CNCF survey, they found that a majority of respondents (66%) are employing Kubernetes to run generative AI workloads. However, they explain further that this majority is composed of respondents that have either fully adopted (23%) or partially adopted (43%) Kubernetes for their inference workloads. The 2024 CNCF survey reported that almost half (48%) of its respondents are not running AI/ML workloads and considered the rest of the respondents (52%) as “early adopters” of Kubernetes for AI/ML containerized app development [13]. In the 2025 survey, they note that respondents that have begun to adopt Kubernetes generally begin doing so for specific use cases, including batch inference jobs or development and staging environments, while maintaining their legacy systems as production environments [33]. They also found that 18% reported that they are planning to adopt Kubernetes for running inference

workloads. They note that these respondents could be facing obstacles that include investments in proprietary ML platforms, concerns about operational complexity, or the need to retrain their teams.

However, they emphasize that most of these respondents are consumers rather than producers of AI models. For example, they found that a majority of respondents (52%) do not build or train AI models. Furthermore, they found that less than half of respondents (47%) deploy generative AI models into production at an occasional frequency (i.e., a few times a year) and only 7% do so daily. They note that although AI models require statistical validation through complex testing of performance and other metrics to ensure production reliability, this slows deployment velocity. Regarding the 7% of respondents achieving daily AI deployments, they note that these respondents have likely implemented automated retraining pipelines that continuously incorporate new data.

In the 2025 CNCF survey, they report on 10 types of AI/ML workloads that respondents are running on Kubernetes: 1) model experimentation (48%), 2) real-time model inferencing (44%), batch jobs for AI/ML pipelines (40%), 3) data preprocessing (40%), 4) batch model inferencing (28%), 5) training large-scale models (24%), 6) data labeling (24%), 7) model monitoring and/or drift detection (16%), 8) hyperparameter tuning (12%), 9) automated retraining (12%), and 10) feature engineering (12%). This builds on the 2024 CNCF survey, which was the first of the CNCF surveys to distinguish among the types of AI/ML development tasks that respondents are working on. However, the 2024 CNCF survey only reported on 5 types of AI/ML workloads that respondents are running on Kubernetes: 1) batch jobs for AI/ML pipelines (11%), 2) model experimentation (10%), 3) real-time inference (10%), 4) dataset preprocessing (9%), and 5) batch model inferencing (8%).

They found that containers are not without their challenges, although the types of challenges have changed since 2023. In 2023, CNCF survey respondents reported security (42%), complexity (38%), and monitoring (37%) were the biggest challenges in using containers [13]. In 2024, CNCF survey respondents reported organizational challenges regarding cultural shifts in

development teams (46%), CI/CD (40%) and a lack of training (38%) were the biggest challenges in using containers [13]. In the 2025 CNCF survey, again, almost half (47%) reported cultural changes with the development team as the biggest challenge in using containers, followed by a lack of training (36%) and security (36%) [33].

They conclude that the 2025 CNCF survey results reflect that as mature cloud native practices have helped make technical challenges (e.g., security, networking, storage, observability) more manageable, organizations are able to focus their attention on culture and development process shifts. They note that cultural resistance may manifest in multiple ways for people with different job roles. For example, developers may believe that containers add unnecessary complexity for simple applications or question whether Kubernetes is production-ready. As well, operations teams (i.e., system admins) may resist what they perceive as developer toys and express concern about troubleshooting containerized systems. Furthermore, management (i.e., managers) may worry about costly distractions from feature delivery and dependencies on specialized knowledge.

The CNCF annual surveys report on the tools developers use that are graduated CNCF projects (i.e., mature and widely adopted). However, the CNCF annual surveys do not report on the development task context in which these tools are used. Although they distinguish among the types of AI/ML development tasks that respondents are working on, they do not report on the challenges associated with conducting these AI/ML workloads on Kubernetes. Furthermore, they do not compare how tools and tasks relate to varying job roles and workloads. In our study, our findings confirm the challenges reported in these prior surveys and provide additional details about the nature of these challenges, as discussed in Chapter 5.

Survey of Kubernetes users finds that they frequently switch among tasks that diverge from their expertise.

In prior work, we conducted an anonymous survey of 29 Kubernetes users from organization A to understand the challenges of targeting distributed batch-processing AI/ML code to cloud resources [80]. We found that the challenges largely stem from the need for every team

member to know everything about the mechanics of achieving scale. As a result, both running and writing code become overwhelming tasks. We found that users switch between 3–7 types of tasks on a daily basis (high multiplicity). Furthermore, the nature of these tasks differ greatly from the users’ role expertise (high divergence). We found that all users assume the daily burdens of cluster operators and frequently take on tasks related to deployment, dependency management, tracking cluster health, and coordinating job submission.

Our study is meant to provide in-depth insights into the breadth of challenges experienced by developers employing Kubernetes for AI/ML containerized apps. Moreover, as we are using interview rather than survey protocol, it provides an opportunity to get more accurate results rather than self-reports. For example, as discussed in Chapter 5, we were able to correlate developer’s job roles to their tasks, which the survey did not.

3.3.2.2 Tool Interventions

We discuss tool interventions that provide relevant context for understanding the results of our study as they present containerized development solutions for developers with either varying levels of experience or job role expertise. Below, we discuss three tool interventions that focus on supporting developers with varying job role expertise [127, 157, 156], two tool interventions that focus on supporting data scientists with varying levels of experience [125, 149], and two tool interventions that more broadly focus on supporting data scientists [129, 164].

Katib framework for hyperparameter tuning supports data scientists and operations engineers.

Understanding the practices of developers with varying job role expertise (i.e., data scientists, operations engineers) that perform hyperparameter (HP) tuning (i.e., fine-tuning) of AI/ML models is related to our study, as we focus on AI/ML containerized app developers employing Kubernetes clusters for purposes related to varying job role expertise and their relevant tasks, including data scientists developing and fine-tuning AI/ML model applications.

George et al. built Katib, a Kubernetes-native framework to address the challenges of

automatic HP tuning, including the different requirements and expectations of developers with varying job role expertise [127]¹. They point out that although several existing frameworks cover the same ground (i.e., running parallel trials and customization search algorithms), Katib provides further support (i.e., multi-tenancy, distributed training, cloud native, and extensibility) and is the only open-source framework that can realistically be run as a hosted service in a production environment. This is, in part, due to the fact that Katib is the first framework that was designed with a focus on two broadly categorized job roles that have different requirements and expectations from a HP tuning framework, shown in Table 3.9.

Table 3.9. Job role, interpretation, and list of primary requirements for users of the Katib framework, built by George et al. [127].

Job Role (Interpretation)	Primary Requirements
Data Scientist (User)	1. Conduct initial experiments with HP tuning in a limited resource environment (e.g., laptop) 2. Once initial results are good, conduct experiments in a production environment (e.g., GPUs, TPUs) 3. Compare and visualize results after HP tuning 4. Track, version, and share experiment details and results
Operations Engineer (Admin)	1. Perform resource efficient deployments 2. Support multiple users in the same cluster with dynamic resource allocation 3. Share the cluster with non-HP tuning jobs 4. Perform capacity planning based on the user workloads 5. Upgrade live system without affecting other users or running workloads 6. Identify issues in the system through logging and monitoring

¹ <https://github.com/kubeflow/katib>. As mentioned in Chapter 1, Katib is part of the Kubeflow project, hosted by the CNCF. The instructions for citing Katib included in the README of the GitHub repository say to cite the arXiv paper provided. We do this as, to our knowledge, this paper is not published elsewhere.

Katib considers both the data scientist, or user, job role (i.e., focus on developing, testing, and maintaining production-ready ML models using all available frameworks and tools in the market), as do all prior frameworks, and the operations engineer, or admin, job role (i.e., focus on managing the underlying infrastructure, maintaining its health, and ensuring that the software on the infrastructure is highly available and up to date), which has been neglected by prior frameworks. They emphasize that in order for a framework to be deployed in a production setup, it needs to consider both categories of job roles. They note that some of the requirements of the operations engineer are not unique to a HP tuning system and are the requirements for operating any scalable, cloud-native service. In fact, the primary requirements described in these two broad categories of job roles are found to overlap in practice, as discussed in Chapter 5.

They demonstrated Katib’s effectiveness in terms of portability, multi-tenancy, scalability, and fault-tolerance with various experiments and simulations. Regarding multi-tenancy, admins can grant extra permissions to users that permit shared access to multiple namespaces. They note that this is useful for users who want to collaborate across teams. They note that for better capacity planning of the deployment cluster, admins define in advance the limit for each user to ensure that cluster resources are not oversubscribed. Resource limits can be set at the individual experiment level, thus restricting the resource usage per experiment.

They also presented several production use cases of the Katib framework for applications in the Ant Financial and Caicloud organizations. At Ant Financial, they found that Katib is paired with the MPI Operator, which helps users obtain the best performance without having to determine the right ratio between the number of worker nodes and server nodes in a parameter server framework. When paired with Katib, users can focus on finding a reasonable HP search space of their chosen model architecture without spending time on tuning the HPs and other downstream infrastructure issues for distributed training. The pairing of Katib and the MPI Operator reflects how organizations use tools as part of a larger ecosystem. However, they do not report any further on the tradeoffs of using these tools together (i.e., how integrated they are). Ant Financial also uses Katib in their application named Dingsunbao, a video mobile app that

allows drivers involved in an accident to claim vehicle insurance in real time. They found that the average damage assessment accuracy is 29.1% higher compared with traditional approaches.

DEEP framework for MLaaS helps data scientists with varying levels of experience deal with large-scale data and share developed models.

Garcia et al. built a framework to address the challenges developers face using cloud services for scientific computing, including data organization as well as sharing and serving ML models [125]². The framework is named “Designing and Enabling E-infrastructures for intensive Processing in Hybrid Data Clouds” (DEEP-Hybrid-DataCloud), which we refer to as DEEP. DEEP provides ML-as-a-Service (MLaaS) to reduce the complexity data scientists face in employing less abstracted cloud services such as IaaS for scientific computing. MLaaS is most similar to PaaS in terms of infrastructure transparency.

They note that with the increasing availability of large amounts of data, scientists face challenges in data organization. For example, they note that since large amounts of data are commonly distributed, users must manage several tasks, including provisioning the necessary computing resources, storage capacity, and orchestrating various infrastructure components in different places. However, they note that such tasks cannot be managed by all users in scientific computing communities. Rather, they note that most users have distinct job role expertise with domain knowledge in specific fields that may not include the technological or infrastructure knowledge necessary to perform such tasks. To bridge these knowledge gaps, they emphasize that cloud service infrastructure providers must abstract the complexity of these tasks and allow scientists to focus on their expertise (i.e., modeling a problem, as well as evaluating and interpreting the results of algorithms). To effectively abstract away these tasks, they emphasize that cloud service providers must understand the requirements of scientific computing communities. Table 3.10 breaks down users with varying levels of ML experience in scientific computing communities into three broad categories.

² <https://deep-hybrid-datacloud.eu>

Table 3.10. Cloud service users in scientific computing communities can be classified into three categories depending on their level of ML experience and goals. Garcia et al. built the DEEP framework with consideration for these user differences [125].

User	Required Level of ML Experience	Goals
Basic	• Almost no ML knowledge	• Use tools that have already been developed (i.e. models) • Apply developed models to their data
Intermediate	• General knowledge on modeling ML problems • Basic programming skills to prepare their data	• Re-train available tools, fine-tuning them to their data • Access to significant computing resources
Advanced	• Competent in ML • Advanced programming skills to design their tools	• Develop ML tools • Access to even more significant computing resources

The general ML development cycle consists of a series of steps (i.e., dataset acquisition followed by iterations of several steps to develop a model, such as model training/testing/evaluation before it is eventually shared/served). They note that, from a computational perspective, the steps of this development cycle are not all equal (e.g., training is more computationally intensive than validation and classification steps). Furthermore, they note that the learning algorithm used also impacts how the distributed computing and computing architecture itself are needed.

Regarding several steps in the general ML development cycle, they note that both sharing and serving a developed model are also challenges that scientists face. They point out that there is no common way to share a developed model as users may require different methods and tools for different models. They note that serving a model over a network has the same challenges as data organization, as this task requires technological knowledge (e.g., API implementation, security) that many scientists may not have. To address these challenges, they presented DEEP, a framework designed to streamline the entire ML development cycle on distributed computing infrastructures (i.e., European Open Science Cloud ecosystem).

DEEP is comprised of four high-level components: 1) the DEEP Open Catalogue, 2) learning facility, 3) DEEP-as-a-Service (DEEPaaS), as well as 4) storage and data services. The DEEP Open Catalogue provides ready to use ML and deep learning (DL) applications and models for users to browse, share, store, and download. The learning facility coordinates and orchestrates model training, testing, and evaluation. DEEPaaS provides a way to serve a developed model, stored in the Open Catalogue. Storage and data services provide storage for user data, training and validation results, as well as any other data assets. Users may either start developing a model from scratch or re-use one of the existing models published in the Open Catalogue. With either approach, the model is encapsulated inside a Docker container with all the required libraries and the DEEPaaS API component. The DEEPaaS API ensures consistency across all the modules that are published into the Open Catalogue and enables the model serving step. All user interaction, locally or remotely, is performed through the DEEPaaS API via a web-interface.

They presented several use cases of the DEEP framework for applications in the image classification of phytoplankton and biodiversity. Regarding phytoplankton classification, DEEP provided scientists from the Belgian LifeWatch Observatory the Flanders Marine Institute (VLIZ) with the ability to deploy an automated classification engine, such that they do not need to deal with infrastructure allocation details, and can perform inference in a fast and scalable manner compared to manual classification done by a taxonomy expert. For this use case, they evaluated DEEP's performance and scalability by conducting 100 consecutive requests to the same endpoint, evaluating the time required to provide a response. They found that the first request took about 45 seconds to complete and subsequent requests completed in less time, always under 3 seconds. The first request accounts for preparation time, as the framework downloads a Docker image into the local registry and spawns it, while subsequent requests are able to reuse the same Docker container, and only reflects the processing time.

In another use case regarding biodiversity classification, they describe a pipeline for several image classifier applications (i.e., plant, seed, and conus) to demonstrate how starting from an existing module from the Open Catalogue, training new classifiers requires only small changes. For example, no time was spent dealing and adapting the DL code. They found that the most time-consuming task when training these new classifiers was gathering and cleaning the new datasets.

Kubernetes platform for ML development supports collaboration among developers with varying levels of ML and systems experience.

Lee et al. built a Kubernetes ML development platform to address the challenges faced by developers with varying levels of ML and systems experience, including overcoming the learning curves experienced by novices and the more advanced, customized tasks performed by experts [149]. They point out that although some organizations have developed ML platforms for developers with varying levels of experience (i.e., Azure Machine Learning Studio from Microsoft [20], SageMaker from Amazon [62], and FloydHub [5]), these platforms lack customization, support a limited number of algorithms, conduct inefficient resource scheduling, and

are complex for managing training jobs.

They note that in order to develop and manage a production-ready ML model, developers often iterate through a series of tasks as part of a general workflow (i.e., data acquisition, data exploration and cleaning, feature engineering, model training, model evaluation, model tuning, model deployment, and model prediction). They emphasize that in order to support developers with varying levels of ML knowledge to more easily complete their workflows, it is important to provide them with an integrated environment that includes the possibility for accessing help from other developers in completing each task. They emphasize that, as a platform for ML developers of varying levels of experience, customization is essential. For example, they note that existing ML frameworks are diverse and each one has certain tradeoffs. Regarding collaboration, they emphasize that communication for problem-solving and instruction are possibilities available through an web ML platform that may reduce learning curves.

They presented a Kubernetes ML model development platform designed to simplify the ML model development process for developers with varying levels of experience. The platform's architecture was composed of five layers, including: 1) the hardware (i.e., physical, virtual), 2) computing cluster (i.e., distributed computing resource and storage clusters), 3) training framework (i.e., support for multiple computing engines and ML libraries to perform tasks of various types and scales), 4) application (i.e., multi-tenant management and collaborative editing, modeling and coding environment, model deployment, asynchronous task scheduling, offline job scheduling), and 5) web-interface (i.e., JupyterLab notebook [47]) layer. Kubernetes is used as the underlying computing resource cluster, with a Docker container runtime and a GlusterFS [37] distributed storage cluster.

The components of the training framework layer include popular ML frameworks and algorithm libraries (i.e., TensorFlow [68], PyTorch [60], and Keras [49]). The hardware resources are hidden from the training framework layer via abstraction of the underlying computing resource layer, such that tasks in this layer can be directly delivered to the computing resource cluster without knowing the specifications of the underlying hardware. As a result, the platform

can be customized to provide more ML frameworks for the training framework layer.

For model deployment, the platform uses OpenFaaS, a serverless model based on Docker and Kubernetes. With OpenFaaS, users do not need to manage the infrastructure of the underlying services and can focus on developing algorithms for ML models. Offline task scheduling is accomplished via RabbitMQ and Celery. They note that JupyterLab notebooks provide the modeling and coding environment that is ideal for the iterative and interactive nature of exploratory data analysis and visualization. As well, Notebooks can be shared to replicate experiments. Multi-tenant collaborative editing of Notebooks is another feature of the platform.

They conducted experiments to evaluate the platform's performance and compared the results to self-hosting VMs running ML environments and jobs on the same hardware. They evaluated environment launching speed and instance capacity, as well as the training performance of different virtualization methods. They found that the platform's virtualization method has a significant advantage in terms of launching speed (i.e., the platform took 2s on average to launch the coding environment and the VMs took 14s) and resource utilization (i.e., memory usage of the platform's Kubernetes pods was 54MB on average and the VMs used about 163MB). Regarding the training performance, they found that the CPU utility and performance among job training of the platform and the VMs to be the same.

They also presented case studies to demonstrate the platform's application in real-world scenarios, including for a temperature prediction model and ML coursework projects. A telecommunications organization used the platform to develop a temperature prediction model as part of their energy saving system. They found that the model's performance was comparable with real temperature data in normal and abnormal periods, practical for production use. Regarding ML coursework projects, the platform supported 300 students over a two week period to develop a garbage classification model. When students encountered problems, they could send an assistance message to experienced users and collaborate in real-time using the Notebook environment by exchanging messages and code. They found that the platform produces well-trained models and preserves the entire modeling process. They note that the deployed model can provide

an intuitive Restful API and a simple web interface that can be used by those without any programming or technical knowledge.

The platform focuses on supporting both inexperienced and experienced ML developers, similar to the DEEP framework [125]. For example, the platform’s collaboration feature allows ML developers with varying levels of experience to work together and provide help to novices from experts when needed. Although they described this collaboration feature in the case study presented for ML coursework projects, they did not provide any details regarding when the students made use of it aside from alluding to debugging bottleneck issues. It is not clear for what tasks or at which point in the ML model development cycle the collaboration feature was used. A deeper understanding of how developers used this collaboration feature could help to further evaluate and improve it. This collaboration feature is particularly salient to our study’s results. Specific AI/ML containerized app development tasks and concepts that developers struggle with understanding and reach out for help with, or that lead to educational materials, presentations, or more comprehensive documentation to be created are discussed in Chapter 5.

OCDL platform for MLOps and DevOps supports collaboration among data scientists, project managers, and application developers.

Liu et al. built OCDL, a combined ML Operations (MLOps) [114] and DevOps platform to address the challenges faced by different teams conducting ML development, including the time-intensive process of collaborating to develop production-ready models [157]. MLOps combines tools, pipelines, as well as processes to efficiently develop ML models and was first introduced by Sculley et al. to address ML-specific risk factors in system design [188]. Liu et al. note that MLOps differs from general DevOps due to how ML applications are developed. For example, they note that ML models are first coded, then are iteratively trained, tuned by HPs and datasets, as well as tested until they are ready to be deployed to a hosting application. They note that the entire process of developing a production-ready ML model can take a long time (i.e., six-18 months), in part, due to requirement changes.

They emphasize that connecting MLOps development and DevOps CI/CD practices can

ease the process of model release (e.g., for sharing and re-use) and deployment. They note that model release processes involve collaboration between developers with varying job role expertise, including data scientists, ML project managers, and application developers. They emphasize that this collaboration introduces complexity in the ML development cycle, as developers with varying job roles make use of their own tools and multiple types of computing resources (i.e., CPUs, GPUs). They point out that although cloud service platforms don't address these varying job roles, they provide MLOps support to help developers manage this complexity. However, they point out that using these platforms can introduce vendor lock-in as they come with their own infrastructure (i.e., computing resources of GPUs, CPUs, storage). They emphasize that an alternative solution is to use an MLOps platform with open-source frameworks that can run on any type of cloud resources.

They presented OCDL, a MLOps platform to support collaboration among developers with varying job roles in the ML application development cycle, automate model deployment for ML applications, and provision data and computing resources for these developers. They define four phases of MLOps, consistent with the general ML application development cycle, that are supported by the OCDL platform. These phases are shown in Table 3.11, with descriptions of the OCDL platform features that support the phases. Each phase is operated by developers with varying job roles that collaborate with each other, as shown in Table 3.12. These job roles provide a context for OCDL to support distinct model operations, cloud resource access, and data management. OCDL connects MLOps development and DevOps CI/CD practices in the release phase. For example, after a data scientist sufficiently tests a developed model's accuracy, they forward the model to their project manager who may then approve the model to be committed to the application repository for release, which begins a series of CI/CD tests (i.e., portability, function, QA, and stability). If the model is not approved, the project manager may send the model back to the data scientist to repeat the process. After all tests have passed, the model is deployed to the ML application.

Table 3.11. Phases of the MLOps cycle that are supported by the OCDL platform built by Liu et al. [157].

Phase of MLOps	OCDL Platform Features
Model Programming	• Web IDE for model coding with templates • Data access to HDFS or online repositories • Choice of GPU and CPU computing instances
Model Training	• Cluster resource provisioning and scheduling managed by Kubernetes • Private GPU cluster nodes or pay-as-you-go instances from public cloud services
Model Assessment	• Same environment as model programming to access data, launch computing resources, and run models
Model Releasing	• Pushing the model to a repository or hosting an application • Automatically launching the model as a service through REST API calls

Table 3.12. Job roles of developers that collaborate with each other in the ML application development cycle and their goals. Liu et al. built the OCDL platform with consideration for the collaboration between and goals of these developers [157].

Job Role	Goals
Data Scientist	• Create and train ML models • Submit trained models to project managers for release approval • Share data and models with other data scientists, data engineers, and project managers
Project Manager	• Perform model assessment before releasing models for deployment • Send back insufficient models to the data scientist for revision
Application Developer	• Consume the models deployed in the application

The OCDL platform architecture makes use of open-source frameworks that connect models, data, cloud resources, and devices for ML and DL applications. It is comprised of three layers, including: 1) the capsule (i.e., web Jupyter IDE for model programming, training, and debugging), 2) rover (i.e., computing resources and data management), and 3) launcher (i.e., handles the builds that release models to ML applications) layer. The capsule layer is deployed as a Kubernetes container which enables the separation of the IDE and underlying computing resources to allow for the customization of hybrid cloud resources for model training and releasing. The rover layer is also managed by Kubernetes and provides hybrid cloud resources (i.e., GPU, CPU) that users can switch between. The rover layer also manages data storage via HDFS, divides into three areas, including: 1) the user space (i.e., dedicated to a user), 2) common space (i.e., shared among project members), and 3) staging (i.e., only accessible by the project manager and stores models submitted by data scientists for release approval) area. The launcher layer builds a workflow for model testing, approval, release, and deployment.

They presented a case study to demonstrate the OCDL platform's application in developing three ML models for source code vulnerability detection. They used the capsule layer's Jupyter IDE to train the three ML models over 17 open-source GitHub projects, storing the source code from these projects on resources provided by the rover layer. The trained models were then evaluated for release and deployment using the workflow provided by the launcher layer and the best performing model of the three was selected to be released and deployed to the ML application. They used this ML application to evaluate the OCDL platform source code itself and found that the models were able to detect 15 out of 109 classes of source code vulnerabilities in minutes time.

Given the focus of their study is to develop OCDL, a combined MLOps and DevOps platform, to our knowledge, they did not conduct any in-depth user studies to understand how data scientists, project managers, and application developers collaborate in practice and further support their platform. They did cite a user study conducted by Erich et al. that conducted interviews with six organizations on the challenges of employing DevOps, one of which developed general

cloud computing systems [118]. Erich et al. found that all organizations were positive about their experiences and only minor problems were encountered while employing DevOps. Although the OCDL platform focuses on supporting three categories of job roles (i.e., data scientists, project managers, application developers), they do not clearly explain the job role of application developers with the same level of detail used to explain data scientist and project manager job roles. Referring to Table 3.12, they simply state that they have the role of consuming models. It is stated that the project manager, upon approving the developed model for release, makes a commit that triggers an automated CI/CD process. Similar to Lee et al.'s Kubernetes development platform [149], features for collaboration are particularly salient to our study's results. The specific AI/ML containerized app development tasks that result in collaboration among developers with varying job roles, either as a result of the limitations enforced by the development process itself or from knowledge gaps, are discussed in Chapter 5.

Kubeflow service for integrating ML development process management supports productivity and collaboration among data scientists.

Golubovic and Rocha built a Kubeflow service to address the challenges faced by different teams of data scientists that make use of a variety of development tools and processes at CERN, including how such development practices impact productivity and collaboration [129]³. Kubeflow is an open-source ML platform built on top of Kubernetes that focuses on supporting Kubernetes-native AI/ML projects [81]. They note that originally, CERN developed a large private cloud based on OpenStack for production use that only offered VMs and the ability to attach virtual storage appliances. They note that this has since expanded to include the ability to provision bare-metal resources, advanced networking features such as security groups and Load Balancing as a Service (LBaaS) [3], as well as a managed Kubernetes service to handle accelerators such as GPUs.

They point out that although ML applications in domains such as high energy physics (HEP) and more have led to an increase of ML libraries and platforms available (i.e., TensorFlow,

³ <https://ml.docs.cern.ch/presentations/>

PyTorch, scikit-learn [63]), how they are used throughout the ML development cycle is not clearly understood. For example, they point out that different data scientist teams within an organization may use their own development processes to transform original, raw data into a trained, production-ready model. They emphasize that this can impact productivity as these different teams face similar challenges in creating their own development processes and in collaborating with one another. They note that creating an integrated service requires a combined effort among developers with expertise in ML tools and algorithms as well as those who are more knowledgeable in managing and scaling infrastructure. They emphasize that by using an integrated service, similar workloads can be aggregated to provide a larger amount of resources to different teams and improve the efficiency of resource usage, which is important for jobs that require scarce resources such as GPUs as well as workloads that fluctuate in resource demands.

They presented a Kubeflow service to integrate the management of the ML development process (i.e., data preparation and analysis as well as large-scale distributed model training and model serving). The main features of the service focus on supporting data preparation and analysis as well as large-scale distributed model training and model serving. Regarding data preparation and analysis, JupyterLab notebooks are used as the web IDE for prototyping data analysis code in Python. Regarding large-scale distributed model training, they note that the sequential workflow of Notebooks can be limiting. They emphasize that Kubeflow pipelines manage workflows and the dependencies among multiple steps in a workflow. For example, they emphasize that a common workflow would be to perform some data retrieval followed by preprocessing, then execute a few variations of model training in parallel and aggregate the results. Kubeflow provides three features to support such workflows, including: 1) a web UI for managing and tracking pipeline runs, 2) an SDK for defining pipelines and components via Python, an engine for scheduling pipelines, and 3) the ability to convert a sequential Notebook workflow into a pipeline via KALE [48], a JupyterLab extension.

They note that Kubeflow also provides Katib, the aforementioned framework for automatic HP tuning, to yield the best model performance. Regarding automatic distributed training,

they note that Kubeflow provides support for ML frameworks (i.e., TensorFlow, PyTorch, and MXNET) and Kubernetes via operators. They note that these operators carry out the required container orchestration and setup of training jobs, the required communication and credentials, as well as data exchange and aggregation after completion. They note that this automation improves the complex and error-prone process of distributed training, as users have far less requirements to specify (i.e., the number of workers they require and the type of resources they require access to). Regarding storage integration, they note that the service provides access to multiple storage backends (i.e., EOS, S3, and CVMFS). Regarding model serving, they note that Knative is used to provide a way to query a trained model, getting outputs for a certain set of inputs.

They presented a use case to demonstrate the service's application in training a fast calorimeter simulation with 3D GANs, a popular use case for DL in HEP. They conducted two training runs, the first done at CERN with a low number of GPUs in a multi-node environment, due to limited availability of resources. The second was done in the Google Cloud Platform (GCP) with access to a larger number of GPUs and TPUs. They found that the performance improvement is close to linear. They also found equivalent results when using TPUs.

They presented a cost analysis of public cloud usage to demonstrate that it can provide a solution for the fluctuating resource demands of training workloads that can scale to large amounts of resources. They found that the cost of the training time of one epoch remains similar when increasing the number of GPUs, although the training time is reduced up to 52 times for 128 GPUs and is almost linear up to 64 GPUs. They found that the best results are achieved using preemptible, rather than reserved, TPUs, which are more than twice as cheap as their GPU equivalent regarding the training time. However, preemptible TPUs are only available up to eight cores, with the reserved instance costs being more expensive. For example, they found that the overall training cost with 128 GPUs is the same as TPUs, and takes only half the time.

Scanflow-K8s framework for MLOps mitigates collaboration between data scientists and data engineers.

Liu et al. built Scanflow-K8s, an MLOps framework to address the challenges faced by different teams conducting ML model development, including the collaboration between teams that train and deploy ML models [156]. They point out that ML model development increasingly involves workflows composed of reproducible steps and executed as containerized pipelines to build and deploy models. However, they also point out that deployed models need to be constantly managed, monitored, and debugged to guarantee their availability, validity, and robustness. For example, they note that a team of data scientists performs repetitive model training workflows that can be automated via AutoML modules and frameworks developed for algorithm selection, model selection, and feature selection to tune HPs. They note that once the model has been trained, a different team of data engineers deploy the production-ready model. As part of their workflow, they note that data engineers manage the robustness of the deployed model and any technical debt, monitor data drift, as well as debug security vulnerabilities.

To mitigate these challenges, they emphasize that containerized ML development workflows could benefit from leveraging flexible and diverse autonomic capabilities. They point out several examples of autonomic computing applications in dynamic contexts, including to enable adaptive learning algorithms for streaming data to manage ML models at the application layer or to reconfigure and restructure workflows at the infrastructure layer. They employ an agentic approach to leverage autonomic computing for ML development workflows to handle such dynamic contexts. They note that the agents manage the robustness and requirements of the model at the application layer along with the quality of services (QoS) and the structure of workflows at the infrastructure layer.

They presented Scanflow-K8s, an MLOps framework to enable the autonomic management of ML development workflows. In previous work, they presented Scanflow, an executor multi-graph framework to manage ML development workflows [94]. They note that Scanflow is redesigned to upgrade the executor nodes to a multi-agent system based on triggers, primitives,

and strategies, as well as being integrated with Kubernetes to enable the autonomic management of ML development workflows. The architecture was composed of two layers, including: 1) an application-controlled and 2) infrastructure-controlled layer, to deal with various types of uncertainties in dynamic contexts.

The application-controlled layer detects application-related changes, such as the requirements from data scientists, end-users, and the workflow context. The infrastructure-controlled layer detects internal and external system contexts, as well as adjusts the workflow executors at run-time according to the predefined rules of the resource manager. Regarding the application-controlled layer, four reactive agents are used to perform adjustments in ML development workflows and manage application-related changes (i.e. tracker, checker, planner, and executor). They note that these agents are the fundamental components to implement autonomic ML workflows and that each agent is an independent computational unit that is able to run actions according to perceived changes in state. Regarding the infrastructure-controlled layer, the deployment and execution of containerized ML development workflows are supported via Kubernetes.

They presented two case studies where they conducted experiments involving image classification to illustrate the features of Scanflow-K8s and evaluate the feasibility and effectiveness of their agent approach for the autonomic management of ML development workflows. They conducted these experiments using a 10-node Kubernetes cluster. Regarding the first case study, they used the MNIST and MNIST-C datasets to train and test an image classification model, respectively. They demonstrated how different teams may use Scanflow-K8s to build and deploy their workflows, as well as how the agents manage the workflows at the application layer while running the workflows in production (i.e., to detect and handle drift anomalies). They found that the initial model trained by the data scientist team had 90% accuracy. As the agents monitored prediction accuracy, they detected anomalies (i.e., data drift) and generated new versions of the model that provided improved accuracy (i.e., 91-93%). They conclude that different teams could leverage Scanflow-K8s to manage ML workflows and how the agents collaborate to debug a drift

anomaly problem and upgrade a new, more accurate model.

Regarding the second case study, they used the MLPerf inference benchmark to test batch and online ML inference for image classification. They demonstrated how Scanflow-K8s can deal with dynamic context and non-functional requirements by taking advantage of the resource manager and via collaboration between application and infrastructure layers. They found that the agents used application-level metrics to configure auto-scaling according to QoS requirements. They also found that the agents tuned the container-level resource and affinity configuration for batch workflows to optimize performance according to workflow type and resource availability. They also found that the agents dealt with service unavailability by redirecting the traffic to a backup service defined at the application-level. They conclude that the agents can take actions to keep the performance and availability of workflows in a multi-layer controlled autonomic architecture.

As part of another study by Liu et al., they further extended Scanflow-K8s to implement fine-grained scheduling for containerized HPC workloads in Kubernetes clusters [155]. They note that as Kubernetes is not optimized for managing HPC workloads and therefore, developers must use additional algorithms and tools to improve performance and leverage containerization and orchestration technologies. They found that their fine-grained scheduling approach yielded performance gains for HPC workloads running on Kubernetes [155]. As they used the HPC Challenge benchmark suite⁴ to conduct their experiments and evaluate performance outcomes rather than an AI/ML workload, to our knowledge, it is unclear whether this extension is intended for AI/ML workloads. We discuss this study here as it is an extension of Scanflow-K8s, a MLOps framework. Furthermore, understanding the particular challenges involved in developing a Kubernetes platform for HPC workloads is relevant to the results of our study, as we focus on AI/ML containerized app developers employing Kubernetes clusters for various purposes, including to develop and deploy applications some had either previously ran, or still run, on HPC systems.

⁴<https://github.com/icl-utk-edu/hpcc>

SnapperML framework for MLOps helps data scientists achieve reproducibility.

Molner et al. built SnapperML, an open-source MLOps framework to address the challenges data scientists face in achieving reproducibility when conducting ML model development and deploying trained models to production environments, including reliably and efficiently maintaining ML models in production [164]⁵. They note that although traditional reproducibility in computer science can be defined at three levels (i.e., computational, empirical, statistical), these levels are not as easily applicable to ML because they overlap and make it difficult to achieve reproducibility of final results. For example, they note that even if a data scientist is using the same code, hardware configuration, and environment, subsequent results may vary from the original results. They point out that although the challenge of achieving reproducibility in ML model development has led to MLOps becoming a new field in industry, it is not as popular in research.

They presented SnapperML, an open-source MLOps framework that provides data scientists with features that support reproducibility, including to efficiently track experiments, log hardware details, and perform stochastic control. Their goal in designing the framework was to avoid code modification as much as possible. They emphasize that they achieved this by defining proper YAML files and via Python default features. For example, they emphasize that if new configuration parameters need to be defined, not having to modify existing code is a solution to avoid introducing bugs or increasing technical debt. They note that YAML files were chosen due to their standardized structure and readability. They note that another benefit of using YAML is the easy access to the contents once parsed in Python. They emphasize that an advantage of employing this experiment configuration is its compatibility with version control software such as Git to ensure reproducibility.

SnapperML's architecture is composed of several layers of modules and tools, integrated through a CLI. The main components include MLflow (i.e., to monitor the training process, to version the model) [51], Optuna (i.e., to automate the search for HPs) [76], and the Ray API

⁵ <https://github.com/SnapperML/SnapperML>

(i.e., to distribute and scale the computing needs for Python workloads) [198]. The CLI is used to conduct experiments via the MLflow engine, which allows users to save information about their experiments (i.e., metrics, parameters, logs). The Ray API is used to efficiently conduct experiments as they note it requires no code changes for distributed applications. Experiments can be monitored and controlled through the web UIs implemented for MLflow and Ray. Optuna requires its own database, which they note saves time when retraining the models and provides many the state-of-the-art algorithms. They note that little effort is required from the user in a local setup or in a remote cluster. They conclude that this allows users to have distributed training, experiment tracking, visualizations, and automatic control over the random seeds in their experiments. A job runner module provided by the CLI calls and integrates the code developed by the data scientist, who needs to call a Python script after configuring the experiment via a YAML file.

3.3.2.3 Takeaways on AI/ML Containerized App Development

These prior works contribute to understanding the challenges of AI/ML containerized app development [13, 80, 127, 125, 149, 157, 129, 156, 164]. In general, they found that Kubernetes is increasingly being employed in the domain of AI for a variety of tasks, including batch jobs for AI/ML pipelines, model experimentation, real-time model inferencing, data preprocessing, and batch model inferencing [33, 13]. They found that as mature cloud native practices have made technical challenges more manageable, organizations are able to focus more on culture and development process shifts, in which novices experience more challenges.

Similar to IaC and general containerized development [133, 120], there is a lack of standard practices and tools used by data scientists for AI/ML containerized apps. To address this challenge, Golubovic and Rocha built a Kubeflow service to integrate ML development tasks, such as data preparation and analysis, as well as large-scale distributed model training and serving [129]. Similar to DevOps and microservices development [211], in our own prior work, we found that teams are overwhelmed with the responsibility to perform tasks across the entire

AI/ML containerized app development cycle, no matter their job role expertise [80]. Similar to calls for automated configuration tools in prior work on general cloud development [210, 207], a Kubernetes framework was built to automate HP tuning that considers the requirements and expectations of both data scientists and operations engineers [127].

A Kubernetes framework [125] and platform [149] were built to support both novice and expert data scientists as well as ML model developers. Regarding MLOps development, Kubernetes frameworks were built to support data scientists [164] as well as mitigate collaboration among separate teams of developers with varying job role expertise (i.e., data scientists, data engineers) [156]. Regarding MLOps and DevOps development, they developed a Kubernetes platform to address the challenges faced by data scientists, project managers, and application developers who all use their own tools and types of computing resources to develop ML models, which results in a time-intensive collaboration process [157]. The platform automates ML model deployment and establishes a standard workflow for data scientists and project managers to iteratively develop and test models, as well as approve sufficiently developed models for deployment to an ML application, respectively.

Regarding employing Kubernetes for HPC app development, Liu et al. further extended Scanflow-K8s to implement fine-grained scheduling for containerized HPC workloads in Kubernetes clusters and take away the burden of improving performance from the user [155]. This burden is not only addressed in tool interventions for MLOps, but also in tool interventions for general containerized development. For example, Dakic et al. built a Kubernetes platform to automate containerized development for HPC workloads and allow users to focus on their main goals (i.e., scheduling jobs) rather than the time-consuming tasks that involve steep learning curves for novices, such as managing storage and networking, writing YAML, building container images, as well as scheduling and monitoring jobs manually [111]. However, to our knowledge, neither of these studies conducted or cited any in-depth user studies to understand the existing challenges faced in general or AI/ML containerized app development for HPC workloads.

AI/ML containerized tool intervention studies lack the support that user studies provide.

Although these tool intervention studies introduced novel platforms [149, 157], frameworks [125, 127, 156, 164], and a Kubeflow service [129] to mitigate the challenges of AI/ML containerized app development, only four included case studies to support their tool intervention [149, 157, 156, 164]. None of them conducted a user study to support their tool intervention as a proof of concept. Our study builds off of these prior tool intervention studies by conducting interviews to understand the perspectives of AI/ML containerized app developers that have a wide range of development experience, including general, cloud, containerized, and AI/ML app development experience. As well, at the time the interviews were conducted, these developers were from two different organizations that were either at the beginning stages of developing a Kubernetes platform or already had a mature platform. With a more comprehensive evaluation of these different perspectives, our study provides a deeper reflection, understanding, and support for the future presented by the tools introduced in these prior studies, and further insight on how such tools can be prioritized and improved.

3.3.3 Takeaways on Containerized Development

Adding to the general consensus reached by prior work on cloud development perspectives, many of these prior works emphasize the challenges that containerized developers face from a lack of sufficient training [13] and point out existing problems with technical documentation [120], such as a lack of usefulness. Specifically, they found that Kubernetes developers would benefit from documentation to better understand how to employ Kubernetes for running their applications in practice or that explained how to build tools that integrate with Kubernetes [120]. As well, these prior works highlight similar, new challenges in the software development cycle that have been introduced by containerized development, including cultural changes [13] as well as changes in development processes that result in a lack of standard practices [125, 120, 157] as well as an increase in the number of tools used [120, 157, 80] and the frequency that developers must switch among them [80]. Furthermore, these prior works

emphasize how these new challenges are compounded by AI/ML containerized app development, despite an increase of abstraction [125] and automation [127, 156, 157]. For example, changes in development processes introduced by the AI/ML model development cycle [125, 149], such as an increase in collaboration among developers with varying job role expertise [211, 156, 157].

Regarding this increase in collaboration and tooling, prior work presents solutions for integrating AI/ML containerized app development processes that support developers with either varying levels of experience [135, 13, 125, 149] or job role expertise [120, 80, 127, 157, 156] perform at their own level of experience or focus on their own expertise, respectively. However, to effectively abstract away tasks for novices and still be able to support more experienced developers, we must deeply understand the requirements of Kubernetes user communities. Prior work mentioned earlier on the tradeoffs between CLIs and GUIs [206, 203, 187] serve as crucial guidance for how we can evaluate the multiplicity of tools used in AI/ML containerized app development to understand how abstraction benefits or hinders development, as well as how to better integrate tools in general such that developers with varying levels of experience and job role expertise are steadily supported.

3.4 Final Remarks on Prior Work

Many of these prior works, both in cloud and containerized development perspectives, recognize that a multitude of experience and expertise is required by cloud and containerized developers, which contributes to insufficient training and steep learning curves. In our study, we focus on finer-grained job roles (i.e., data scientists, cluster operators, infrastructure developers, library developers, and project managers) and tasks (i.e., dataset management, debugging, image building, monitoring, deployment logic, job submission, cluster observation, application source code, and application versioning) than prior work reports on. In general, the broader categories of job roles focused on in prior work help contextualize and substantiate our own analysis of the challenges faced by developers performing tasks that diverge from their job role expertise.

We also interview developers with varying levels of general software development and AI/ML containerized app development experience (i.e., novice, intermediate, and expert). We combine this factor along with job role expertise to more deeply understand the challenges faced by developers employing Kubernetes for AI/ML containerized apps.

Importantly, there is a missing piece among prior work on understanding the challenges faced by developers employing Kubernetes for AI/ML containerized apps. Given that the focus of the Kubernetes tool interventions presented in prior work on general and AI/ML containerized app development was to build a solution, to our knowledge, none of them conducted any in-depth user studies to understand how their tools are used in practice and further support their solution. To fill this gap, we contribute an in-depth user study to determine whether the results found in these prior works do or don't generalize. We provide further insight from conducting interviews with developers to understand what they actually do and what really troubles them, in practice. Based on our interviews, we address what parts of these tools would be useful and how they could be further improved in Chapter 6.

Chapter 4

Methods

Among other things, the purpose of conducting interviews is to uncover the meaning behind, and the relationships within, people’s work. Such insights can, for example, contribute to design for better tool support. There are two critical elements that go into an interview study: the interview method and the analysis method. The choice of each is dictated by the nature of the topic and the research questions. Our primary research question and hypothesis are:

RQ. As people in larger teams often have distinct job roles with specialized task responsibilities that are intrinsic to those roles, to what extent does AI/ML containerized app development impact the separation of responsibilities among roles?

H. We hypothesize that AI/ML containerized app development, even as orchestrated through Kubernetes, undermines the separation of responsibilities among roles, leading to challenges due to a divergence from specialized tasks that requires developers to perform a multiplicity of new tasks and use tools outside of their specialization, resulting in increased communication with others for help.

Due to the complexity of our topic – the issues encountered in AI/ML containerized app development – we conducted *contextual* interviews. Contextual interviews are part of the Contextual Design methodology [141], and involve interviewing participants in their work context so that they can illustrate their interview responses with their work products, tools,

processes, and other affordances of their work setting [122]. Furthermore, although interviews generally reveal what people *think* about a subject, prompting the interviewee to show relevant examples and demonstrate what they mean provides support for their beliefs.

As for the analysis method, there are two important characteristics for our study. For one, software development is a mature field that has been studied extensively. Thus, it would behoove us to use analytical frameworks from this field in a top-down fashion. At a minimum, this allows us to compare our results to others in software development. On the other hand, there are novel aspects to our research question at the intersection of AI/ML and containerized application development that may not fit historical frameworks. This requires a more open, bottom-up, analytical perspective. We employ each in conjunction, as discussed in the following.

4.1 Top-Down Perspectives

4.1.1 Work Partitioning According to Specialization

As our research question centers on the challenges of AI/ML containerized app development in a team setting, the influence of job specialization on the relative difficulty of development is an obvious intervening variable.

In general, organizations define job roles to help people specialize within a team to maximize productivity, and this is true in many software development settings. A classic example, recalled in many software engineering courses, is Henry Ford’s moving assembly line that combined the ideas of interchangeable parts, subdivided labor, and the fluid movement of materials to improve manufacturing efficiency [200]. Related to the concepts presented, many software development organizations strive to follow principles of modularity, code reusability, and scalability (interchangeable parts), as exemplified by David Parnas’ fundamental system design research paper “On the Criteria To Be Used in Decomposing Systems into Modules” [173].

Agile development teams emphasize specialized job roles that focus on the specific components of an application’s frontend, backend, and infrastructure (subdivided labor) [175].

Tools such as GitHub enable development teams to work on code concurrently to be merged into a cohesive product, while also providing a centralized source for documentation and communication that is frequently integrated with team messaging channels like Slack (fluid movement of materials) [36].

Many software development organizations have specialized job roles as there is a cost to performing work that diverges from one's specialization. For example, there is a learning curve that takes time and training to overcome when a novice is developing new skills. Furthermore, there is a cost to coordinating with others whose expertise diverges from one's own specialization. The time an expert spends training a novice will partially depend on the novice's background. For example, what the novice already knows or what existing misconceptions they have need to be determined by the expert to provide sufficient training. Therefore, the extent of the practice in which work is divided according to one's specialization depends on a person's ability to communicate with experts and access information to develop their knowledge. This practice and its limitations have been well understood for centuries, as introduced in Adam Smith's fundamental economics book, "The Wealth of Nations" [196].

In Chapter 3 we discussed prior work that focuses on specialization, such as between-specialization coordination among developers with varying job role expertise [80, 156, 211, 127, 157, 120, 104] and within-specialization coordination among developers with the same job role expertise [129, 149, 194]. In our study, we explicitly focus on between-specialization coordination in our analysis. For example, a data scientist and cluster operator working on different teams or projects likely do not have a shared vocabulary and understanding of their tasks. This is in contrast to within-specialization coordination. For example, a team of data scientists that communicate with one another on the same project likely have a shared vocabulary and understanding of their tasks.

4.1.2 Gulfs of Execution and Evaluation

From an ecological perspective, humans are not the only specialists in a project team. Their sophisticated tools are collaborators of a sort, to which tedious, complex work is delegated, but not at zero cost to the human user.

In the 1980's, the paper "Direct Manipulation Interfaces" introduced the concepts of *Gulfs of Execution* and *Gulfs of Evaluation* to help understand the efficacy of a user interface [142]. Since the UCSD methodology was formulated around the challenges of designing complex computer-based tools for IT work, this perspective is highly relevant to modern software development, in which numerous tools are employed as an ecosystem. Not only in the past has prior work used these concepts as part of their methodology, indeed, they are still being used in recent prior work on LLM tool modalities for image generation tasks [180] and conversational interactions with LLMs for general software development [185].

Developers encounter challenges when performing tasks that generally fall into two categories known as gulfs of execution and evaluation. A gulf of execution is defined as the difficulty of achieving the user's intentions with the available actions. One measure of this is how well the system allows the user to do what they intended directly, without additional effort. For example, is it clear what action should be performed next, or is it unclear? A gulf of evaluation is defined as the amount of effort the user must exert to determine the physical state of the system and to determine how well expectations and intentions have been met. One measure of this is how well the system provides information about its state in a form that is easy to acquire, is easy to interpret, and matches the way the user thinks of the system. For example, was the outcome of performing an action expected, or was the outcome unexpected? In real organizations with teams of developers that have varying expertise, these gulfs are much more subtle than they appear, as discussed in Chapter 5.

Similar concepts such as Nielsen's 10 usability heuristics have also remained relevant and unchanged since their introduction in the 1990's. Chapter 3.1 defines Nielsen's 10 us-

ability heuristics along with other prior work on tool design and usability that demonstrated how developers choose tool modalities based on their task context rather than their level of experience. Furthermore, AI/ML containerized app developers may switch among a combination of specialized tools with different modalities to accomplish their tasks. Although Nielsen’s 10 usability heuristics provide more fine-grained guidance for evaluating the design of tool interfaces, the gulfs of execution and evaluation provide a better lens for understanding what AI/ML containerized app developers themselves actually do and what problems they encounter, in practice.

4.1.3 Defining the Relationship Between Directness and Modularity

In Chapter 3, we discussed prior work on tool design and usability that revealed Although tools may be designed for developers with varying levels of experience, in practice, developers choose the modalities based on their task context rather than their level of experience. Different tool modalities provide users with interfaces that have varying levels of direct-manipulation, a term hatched by Shneiderman [192, 193, 191]. Direct-manipulation systems have three properties: 1) continuous representation of the relevant object, 2) physical actions or labeled buttons instead of complex syntax, and 3) fast, incremental, and reversible operations whose impact on the relevant object is immediately visible [193]. Shneiderman suggests that such systems provide benefits to users with varying levels of experience, as shown in Table 4.1.

Table 4.1. Summary of the benefits of direct-manipulation systems as defined by Shneiderman [193].

Benefits of Direct-Manipulation Systems
• Novices can quickly learn basic functionality, usually via demonstration by an experienced user. • Experts can work quickly to carry out a wide range of tasks (e.g., defining new features). • Knowledgeable, infrequent users can retain operational concepts. • Error messages are rarely needed. • Users can immediately see if their actions are furthering their goals and easily adjust their activity.

Ultimately, the benefits of systems with direct-manipulation interfaces come with trade-

offs¹. Hutchins et al. introduced the gulfs of execution and evaluation as an analysis framework for measuring the tradeoffs of directness in system interfaces [142]. They describe “directness” itself as an impression or feeling regarding an interface [142]. Directness can also be described in terms of what a user experiences when interacting with a system and thus we refer to it as an experience from here on. They suggest that such an experience of directness is inversely proportional to the amount of cognitive effort it takes to manipulate and evaluate a system and moreover, that cognitive effort is a direct result of these gulfs [142]. Furthermore, they state that directness is always relative, as it is often due to the interaction of multiple factors [142].

The experience of directness results from two distinct factors: 1) distance and 2) engagement [142]. Distance refers to the gap between a user’s thoughts and the physical requirements of the system they are using. For example, a short distance means that a user’s thoughts are readily translated, in a straightforward manner, into the physical actions required by the system and that its output is in a manner readily perceived and interpreted in terms of the user’s goals. A system’s interface introduces distance to the extent that there are gulfs between a user’s goals and knowledge as well as the level of information provided by the systems with which the user must work with. The goal of bridging these gulfs in user interface design is to minimize the user’s cognitive effort. Gulfs of execution are bridged by making the commands and processes of the system match the thoughts and goals of the user. Gulfs of evaluation are bridged by making the system output present a conceptual model of the system that is readily perceived and interpreted by the user.

Engagement refers to the qualitative experience that a user is directly manipulating relevant objects. Hutchins et al. state that in general, system interfaces have been built on two core metaphors that represent the nature of HCI: 1) the conversation metaphor and 2) the model-world metaphor [142]. Regarding a system built on the conversation metaphor (e.g., CLIs), the interface provides a language medium in which the user and system “talk” in a

¹Although this discussion is about direct-manipulation interfaces, directness applies to all interfaces as do the tradeoffs that come along with it.

conversation about an assumed world that is not explicitly represented. Regarding a system built on the model-world metaphor (e.g., GUIs, IDEs), the interface itself is non-conversational and instead, explicitly represents a world where the user can perform actions on that representation which in response, change the state of the interface. Although conversational interfaces have an implied intermediary between the user and the world, non-conversational interfaces have an explicit intermediary such that the user experiences there is no intermediary, and thus can be considered to have direct engagement.

The directness of an interface may also change as a user performs different tasks. For example, Although a data scientist is able to use the `kubectl` CLI to launch a AI/ML job in a Kubernetes environment, they are not able to easily monitor it. To more easily monitor their job, they may use a GUI such as a Grafana dashboard that provides visualizations and other information about the cluster. Therefore, Hutchins et al. [142] state that an analysis of the nature of the task being performed is essential in determining the directness of an interface.

Importantly, directness is not a property of the system's interface alone and implies an important relationship between the task the user is focused on as well as the way that task can be accomplished via the input and output interface languages of the tools in their ecosystem. To expand on the notions of distance above, Hutchins et al. describe two properties that lie between the user's conception of the task and the interface languages: 1) semantic distance and 2) articulatory distance [142]. As stated in Chapter 4.1.2, the concepts of the gulfs of execution and evaluation have been used in past prior work as well as in recent prior work. The concepts of semantic and articulatory distance have also been used in recent prior work on test generation tools [112], VR-mediated communication [183], and vibe coding [161].

At the highest level of information, a task may be described by the user's intention: "write a C program that prints the message 'Hello World!'" At the lowest level of information, performing this task consists of manipulating bits inside a computer system. The intermediary between the higher-level system interface and the lower-level operations of the computer is the system's task-support structure that implements these expressions of information in the

interface's language [142]. For example, in introductory C programming language courses, students learn about the binary representations of numbers and characters, manually performing these conversions to gain a deeper understanding of the underlying system's representation of the world. Suppose that the C programming language did not implement an automatic conversion of the user's input of numbers and characters for them and instead, its interface language was of minimal complexity, being near or at the level of bit manipulation. Then, the user would carry the entire burden of bridging the gulf between their high-level intention and the low-level representation of the instructions given to the system. In doing so, the user must provide their own task-support structure, extensively planning out the instructions themselves for performing additional conversions, which is challenging and introduces more opportunities for error.

Semantic distance refers to the meaning of an expression in the interface language in relation to what the user wants to say [142]. For example, is it possible for the user to say what they want to in the language? Therefore, does the language support the user's conception of the task domain? Furthermore, can the user say what they want to in a concise, straightforward manner, or must the user construct a complicated expression in relation to what they think should be a simple task? Therefore, semantic directness results from matching the level of information, high or low, required by the interface language to the level at which the user thinks of the task. Semantic distance in the gulf of execution represents how much of the required task-support structure is provided by the system and how much by the user. The distance increases and the execution gulf widens when the user must provide more structure than the system with regards to specifying the meaning of the input expression used to achieve their goal. Semantic distance in the gulf of evaluation represents how much task-support structure is required for the user to determine whether their goal has been met. The distance increases and the evaluation gulf widens when the user must translate the output from terms that are not related to their goal to those that are compatible with their goal.

For example, suppose a user's goal is to run a job using the kubectl CLI and verify that it completed. The user first runs a command to apply their configuration file (i.e., `kubectl apply`

`-f job.yaml`), then observes the output that their job was created (i.e., `job.batch created`). To verify the completion of their job, they run another command to view the completed pods of their job (i.e., `kubectl get pods`), however they do not observe any output. They must repeatedly run this command over some time and wait until they finally receive an output with the id of the completed pod of their job (i.e., `job-5rwd7`). The information needed for evaluation is finally presented in the output, yet it required the user to repeatedly provide information to the system. Moreover, when the output provided no information, this was not in a form that directly fits the terms of evaluation. The burden is on the user to perform this repeated work and to understand that no output means their job is not yet complete, which requires more effort. Suppose viewing the status of the completed job was more directly achieved with the command to view the completed pods triggering a script that automated the process such that upon an initial output of nothing, it did not require the user to repeatedly execute the command and instead, continued to periodically run the command automatically and provide intermediate status updates until a final output was given. Such a process would reduce the mental workload and shorten the semantic distance between the user's goal and the interface output language.

From a different perspective, tasks that are sufficiently practiced by users become, in a way, automated such that they require less cognitive effort. However, this does not reduce the semantic distance because the user alone must overcome the gulfs between their intentions and the interface, through sufficient practice.

Hutchins et al. state that in general, there are two basic ways to reduce semantic distance: 1) on the system side, requiring effort on the part of the system designer, and 2) on the user side, requiring effort on the part of the user [142]. Regarding the system designer, they can create higher-level, specialized languages in order to make the semantics of the interface input and output languages more directly match the user's intentions. Ideally, specialized languages close the gap between a user's intention and the interaction language while preserving their freedom and ease of general expression by allowing for extensibility of the language. In our study, we assume that our participants are using well-designed tools. Therefore, we focus on the user side.

Regarding the user, they can develop expertise by learning to think in the same language as that required by the system in order to bridge the gulfs. However, there are tradeoffs between the costs to the user of learning a new way to think about a domain and the potential added power that comes along with it. For example, changing the users' conception of the domain may prevent some intentions from arising at all. Therefore, although a well-designed, specialized language may give the user a powerful way of thinking about the domain, it may also restrict their flexibility to think about the domain in different ways.

A third solution to reduce semantic distance combines these two perspectives: users can also act as system designers by creating their own custom tools, which work in continuity with how they already think. Custom tools are created by users to abstract the more cumbersome tasks away such that they are able to either avoid or more quickly overcome the learning curve that comes with those tasks. Furthermore, these custom tools can be shared with other users on their team so that they also benefit from their use, even without having to invest their own time and effort in creating that tool.

Articulatory distance refers to the meanings of expressions in relation to their physical form [142]. For example, on the input side, the form may be keyboard-presses for a CLI or mouse-navigation and clicks for a GUI. Likewise, on the output side, the form may be a string of characters for a CLI or an updated icon in a GUI. Maintaining direct engagement can be accomplished through the form and speed of feedback. Readily available, quick feedback allows for the modification of actions even as they are being executed and removes the perception of an intermediary by providing a continuous representation of the system's state. As well, if a user is able to watch the actions take place and monitor them as they happen, they are able to evaluate the outcome of those actions more quickly and with more understanding. This way, the experience of direct engagement results from a reduction in the cognitive load of maintaining relevant information and the form of the interaction. Hutchins et al. state the requirements of a system that provides the experience of direct engagement, shown in Table 4.2.

Our results contribute to understanding how the concept of directness in HCI and the

Table 4.2. Summary of the requirements of a system that provides the experience of direct engagement, as defined by Hutchins et al. [142].

System Requirements for Direct Engagement
• Execution and evaluation should exhibit both semantic and articulatory directness. • To create the experience that one is directly manipulating relevant objects, the interface’s input and output languages should be inter-referential, such that an input expression should incorporate or make use of a previous output expression. • The system should be responsive, with no delays between execution and the outputs, except where those delays are appropriate for the knowledge domain itself. • The interface should be unobtrusive, such that it is not noticed and does not take away from direct engagement.

principles of modularity in software development overlap. In an ideal setting, modularity maximizes directness and minimizes the gulfs of execution and evaluation. For example, a data scientist may be an expert at writing a Python program to efficiently filter a large dataset using the Jupyter notebook IDE. Such a data scientist applies minimal cognitive effort in performing this task, as they know the commands and processes required by the system to achieve their goal, being able to bridge the distance of the gulf of execution. As well, they apply minimal cognitive effort in interpreting the outcome of this task, as they use a Jupyter notebook to readily check and debug the results of their program incrementally, being able to bridge the distance of the gulf of evaluation. Their choice of using a Jupyter notebook provides them with the experience of direct engagement with this task. Furthermore, the semantic distance they experience in performing and interpreting this task is bridged via their expertise and training as a data scientist, serving as the foundation that houses the arsenal of tools they articulate with ease within their own ecosystem.

However, as discussed earlier, there are several solutions to bridging semantic distance to achieve the experience of directness. Using the analysis framework of the gulfs of execution and evaluation, we were prepared to find that directness is relative to a software developers job role expertise and the tasks they perform. The notion that directness is relative is consistent with Hutchins et al. [142]. For example, a data scientist may not be an expert at submitting an AI/ML job in a Kubernetes environment using the kubectl CLI. Perhaps they have not been trained to do

so, or haven't had enough practice, and thus are not familiar with doing so. To bridge the gulfs they face in performing this task, they may require additional training, help from an expert, or a custom tool that either they or someone else on their team created.

As previously stated in Chapter 4.1.1, many software development organizations have specialized job roles as there is a cost to performing work that diverges from one's specialization. Indeed, this is what we mean by modularity: although a data scientist can independently accomplish some tasks due to their specialization in which they have gained expertise, their ability to accomplish other tasks outside of their specialization is contingent on bridging the distance of the gulfs they experience from the system side, the user side, or a combination of the two. As can be understood from the argument above, modularity and specialization, or expertise, are in some sense, the same thing.

To consider the possible solutions to the challenges that different teams of developers encounter in employing Kubernetes for AI/ML containerized apps, it is critical in our analysis to understand to what extent job role expertise and familiarity, as well as the tools and processes used, modulate the gulfs. We further break down the challenges we categorized as gulfs, examining in depth to what extent they impact the experiences of distance and engagement that a developer experiences when performing a task within as well as outside of their expertise and familiarity. We expect to find that in general, a developer's expertise impacts their experience of distance in performing a task, as shown in Table 4.3. In our results, we consider these factors in answering two important questions to illuminate the relationship between modularity and directness: 1) what tasks must a developer perform either within or outside of their job role expertise?, and 2) which tasks are challenging for a developer to independently perform?

Table 4.3. Considering the experience of distance in terms of a developer's expertise in performing a task, the greater their expertise in performing a task, the less distance they experience.

	Task	
Expertise	Job Submission	AI/ML App Source Code
Cluster Operator	High	Low
Data Scientist	Low	High

4.2 Bottom-Up Perspective: Straussian Grounded Theory

Although the theories of gulfs and specialized labor provide a powerful analytical lens, we could not presume to know, in advance, how our participants' narratives, experiences, and work would or wouldn't either flesh out or fit into these concepts. For example, how does someone on a project conceptualize their role, and their responsibilities within that role? This is where bottom-up analysis comes in. We conducted a literature review of interview studies that focus on software developers and found Straussian Grounded Theory [108] to be the most commonly applied variant of Grounded Theory analysis frameworks [116, 211, 187, 189, 205, 145]. In general, Grounded Theory analysis frameworks are used to answer more exploratory research questions, such as ours [128].

Following the techniques of Straussian Grounded Theory, we conducted bottom-up coding of the interview data to develop new codes beyond the two existing gulfs codes. The interview transcripts were coded in three stages (i.e., open-, axial-, and selective-coding) to iteratively categorize the participants' statements. As part of the open coding process, we applied the gulfs to the statements made by our participants and defined them as the challenges that they reported experiencing and further categorized them into a challenge of either interpretation (evaluation) or taking action (execution). During the axial coding process, open codes were categorized into a broad narrative (i.e., conditions, actions and interactions, as well as consequences and outcomes). During the selective coding process, open codes were categorized into a specific narrative of development processes (i.e., application versioning, dataset management, job submission, and onboarding) composed of various stages (i.e., learning, inventing, inner-loop, and sharing).

4.3 Participants

We interviewed 35 developers from two distinct organizations, one industrial and the other academic. Tables 4.4, 4.5, 4.6, and 4.7 provide a more detailed breakdown of our participants. Participants were recruited from industry partners and our personal network in an open call to

developers using email, social media (i.e., Slack), and word of mouth. To gain exposure to a broader range of Kubernetes development experiences, we recruited participants from a larger organization (over 100 employees) and a smaller organization (under 10 employees). Participants had a broad range of job roles and experience using Kubernetes for AI/ML containerized apps. Furthermore, at the time the interviews were conducted, the larger organization was in the process of employing Kubernetes, while the smaller organization had a more mature, established use of Kubernetes. Overall, we recruited 35 participants, all software developers with 2 to 25 years of professional development experience (total of 12 years on average). Our 35 participants came from 2 different organizations – 29 worked in a larger industrial organization and 6 worked in a smaller academic organization.

Table 4.4. Breakdown of the number of participants from each organization, IDs used to represent them, and the average software development experience for each organization.

Organization	Total (Number of Participants)	Participant IDs	Experience (Average)
Organization A	29	A1–A29	12 years
Organization B	6	B1–B6	11 years

Table 4.5. Participant job roles from both organizations, total number of participants within each job role, and average software development experience.

Job Role (Both Organizations)	Total (Number of Participants)	Experience (Average)
Cluster Operator	4	18 years
Data Scientist	13	12 years
Infrastructure Developer	10	11 years
Library Developer	2	9 years
Project Manager	6	9 years
All Job Roles	35	12 years

Table 4.8 provides a more detailed breakdown of our participants’ job roles. We apply higher-level interpretations to these job roles, based on how we defined them in our initial survey, presented in prior work by Arroyo et al. [80] and discussed in Chapter 3.3.2.1. These higher-level interpretations are shown in Table 4.8 and include: 1) Developer, 2) Operator, 3) Scientist, and 4) Other. We map the job roles to these four categories as indications of what participants’ expertise offers to their team.

Table 4.6. Participant job roles from organization A, total number of participants within each job role, and average software development experience.

Job Role (Organization A)	Total (Number of Participants)	Experience (Average)
Cluster Operator	2	20 years
Data Scientist	12	11 years
Infrastructure Developer	8	13 years
Library Developer	2	9 years
Project Manager	5	10 years
All Job Roles	29	12 years

Table 4.7. Participant job roles from organization B, total number of participants within each job role, and average software development experience.

Job Role (Organization B)	Total (Number of Participants)	Experience (Average)
Cluster Operator	2	15 years
Data Scientist	1	20 years
Infrastructure Developer	2	5 years
Library Developer	0	NA
Project Manager	1	5 years
All Job Roles	6	11 years

Participants from both organizations conducted a broad range of development tasks. Table 4.9 provides a more detailed breakdown of our participants' tasks within their job role based on how we defined them in our initial survey, presented in prior work by Arroyo et al. [80] and discussed in Chapter 3.3.2.1. We map the tasks to the same higher-level interpretations shown in Table 4.8 as indications of what participants actually do. For example, on a daily basis, data scientists are working on authoring AI/ML app source code.

Table 4.8. Summary of participants’ job roles and their definitions, along with the higher-level interpretations we applied.

Job Role (Interpretation)	Definition
Infrastructure Developer (Developer Role)	Working on cluster infrastructure (e.g., tools to support cluster operators and users)
Library Developer (Developer Role)	Working on library development for AI/ML (e.g., PyTorch)
Cluster Operator (Operator Role)	Working on cluster management (e.g., giving users access to the cluster)
Data Scientist (Scientist Role)	Working on AI/ML models
Project Manager (Other Role)	Overseeing development projects and teams

Table 4.9. Summary of participants’ development tasks and their definitions, along with the higher-level interpretations we applied.

Development Task (Interpretation)	Definition
Dataset Management (Developer Task)	Curating and managing datasets
Debugging (Developer Task)	Debugging an AI/ML run to determine why it’s misbehaving
Image Building (Developer Task)	Building images (e.g., managing dependencies)
Monitoring (Developer Task)	Monitoring the progress of a run
YAML/Deployment Logic (Developer Task)	Authoring or editing YAML (e.g., forms of deployment logic)
Job Submission (Operator Task)	Arranging a job for submission (e.g., sizing the run, coordinating for time on the cluster)
Cluster Observation (Operator Task)	Observing cluster-wide health (e.g., utilization logs and metrics)
AI/ML Application Source Code (Scientist Task)	Authoring AI/ML source code (e.g., PyTorch, Ray)
Application Versioning (Other Task)	Versioning related to code dependencies (e.g., propagating and testing a new application/version, using an automated or manual release process)

Some of the tasks shown in Table 4.9 may apply more broadly. For example, given their job role, a project manager may have a broader range of expertise and perform many different tasks. Therefore, this higher-level interpretation of their role and the tasks they perform within it helps to indicate what their job role expertise offers to their team and what tasks they actually do.

At the time of the interviews, both organizations were running their own Kubernetes clusters for AI/ML containerized apps. Organization A ran their cluster in an industry research and development setting. Organization B ran their cluster in a university research and development setting. We interviewed the developers that lead the operation of the Kubernetes clusters from both organizations, along with varying levels of users (e.g., namespace users, namespace managers). At organization A, participants were developing AI/ML containerized apps for foundation models related to large language models (LLMs) and conducted model pretraining, fine-tuning, and inferencing. Participants at organization A also conducted spatial-temporal analysis related to geospatial projects. At organization B, participants were developing computer vision (CV) applications related to virtual reality (VR) projects.

At organization A, we interviewed developers from different functional teams, working under the umbrella of same project. These functional teams were composed of specializations such as a team of data scientists and another team of infrastructure developers. These functional teams performed different tasks for the same project that involved developing AI/ML containerized apps on a Kubernetes cluster. For example, while a team of data scientists focused on developing AI/ML app source code for a model, a team of infrastructure developers made improvements to the Kubernetes cluster itself. At organization B, we also interviewed developers from different functional teams working under the umbrella of the same project, CV application development on a Kubernetes cluster. As well, we interviewed developers from organization B that were users of the Kubernetes cluster.

4.4 Data Collection

We conducted semi-structured interviews with developers that had previously used or maintained a Kubernetes cluster for AI/ML containerized apps. For the interviews, we defined a set of 25 questions based on our research questions, shown in Appendix A. Our questions covered the topics of development background and experience, processes, tools, and a walk-through demonstration of developers' typical, everyday work. In the interviews, we covered all questions with each developer and prompted them to show examples of their work. The concrete order of questions varied in a semi-structured manner.

Interviews were conducted in a private setting where the discussion could not be observed or overheard by others. As I was the only neutral third party to both organizations, I alone conducted the interviews. The interviews were conducted either in-person, in the familiar context of a participant's office or a meeting room on-site of their work place, or remotely via Webex or Zoom. Interviews were scheduled to last 45 minutes, conducted in English, as well as audio and video-recorded to capture the conversation and the participant's computer screen when sharing examples of and demonstrating their work. An exempt IRB was obtained in order to collect the audio and video recordings. All participants signed consent forms to agree to being recorded.

4.5 Coding Process

After the interviews, all audio and video-recordings were transcribed. Audio was initially transcribed automatically via Webex or Zoom, then manually corrected and further improved to add context in moments where participants shared their screen to provide examples of their work. The interview transcripts were anonymized by removing any personal information (e.g., names, locations) and replacing internal organization project names with general descriptions. A spreadsheet was used to organize and code the anonymized interview transcripts.

Following the recommendations by Sedano et al. and Campbell et al. [189, 100], I alone, as a neutral third party to both organizations, individually coded the anonymized interview tran-

scripts and met regularly with my collaborators to go over their progress. During these meetings, we would go over any new or changed codes to reach an agreement on their interpretation. This iterative coding process resulted in a final codebook, shown in Appendix C. Following Cito et al. [104], this codebook produced a set of findings that are supported by the statements of multiple participants.

4.5.1 Inter-Coder Reliability (ICR)

The data coding process adopted in many interview studies includes generating an Inter-Coder Reliability (ICR) metric as a result of multiple researchers individually coding the data and comparing their codes to measure the agreement on and consistency of their interpretation [170]. ICR is a numerical measure of agreement between multiple coders regarding how the same data should be coded and is frequently conflated with inter-rater reliability (IRR) [170]. However, IRR is technically used for data rated on some ordinal or interval scale [170, 116]. The process of generating an ICR metric in a Grounded Theory study varies and depends on several, limiting factors shown in Table 4.10.

Table 4.10. Diaz et al. present several limiting factors of generating an ICR metric in Grounded Theory studies [116].

Limiting Factor	Definition
Controversiality	The more potentially controversial the judgment, the more multiple raters are needed.
Practicality	The less practical it is to have multiple raters, the more reasonable a single-rater design becomes.
Philosophy	Involving multiple raters is more important from a realist ontological perspective than from an idealist ontological perspective.

Regarding controversiality, as defined in the first row of Table 4.10, our coding is mostly straightforward (e.g., job role, experience) and requires some level of interpretation (e.g., gulfs of execution and evaluation). Regarding practicality, as defined in the second row of Table 4.10, two of the researchers collaborating on our study could identify one or more participants based on their statements even after anonymizing the transcripts due to close collaborations or other conflicts of interest and recused themselves from individually coding the data. Therefore, I alone acted as a neutral third party to code the data.

Regarding philosophy, as defined in the last row of Table 4.10, a realist ontological perspective assumes that reality can be objectively studied [181]. An idealist ontological perspective builds on this and assumes that an objective, observable reality exists while acknowledging that attempts to study it are influenced by human understanding [181]. The original philosophical perspective taken by Straussian Grounded Theory, which we follow in this study, is not strictly defined, having evolved from a more realist ontological perspective towards assuming that reality is socially interpreted and that understanding this is important to comprehend human behavior [116, 181, 108].

Some qualitative researchers may object that generating IRA or IRR is based on a set of codes derived from initial hypotheses and insights prior to coding rather than a less positivist approach based on Grounded Theory where the codes emerge inductively as themes are identified during coding [116, 101, 128]. Assessing IRA or IRR under these conditions is challenging and research has yet to address this [79, 176]. This is why IRA or IRR are usually performed for text analysis only when a preconceived coding scheme with a fixed number of codes is being used and not when a Grounded Theory approach is being used [139, 124].

Despite these limiting factors (i.e., controversiality, practicality, and philosophy) and coding techniques (i.e., established before or emerged after data collection), Grounded Theory interview studies have generated IRA or IRR via weekly meetings between the one researcher that individually coded the data and their collaborators where they discuss the new codes, any changes to existing codes, and reach an agreement on their interpretation [189]. However, due to the additional limitations of our study, as discussed in Chapter 4.4, we took a different approach following other Grounded Theory interview studies conducted with software developers.

Chapter 5

Results

5.1 Job Role Versus Task Performance

Our initial survey, presented in prior work by Arroyo et al., provided the insight that employing Kubernetes for AI/ML containerized app development often leads developers to perform tasks outside their job roles [80]. What that prior work didn't reveal is how this task divergence breaks down by job role and what challenges are encountered when performing tasks that diverge from job role expertise. Figure 5.1 fills part of this gap by providing a matrix visualization of, for our participants, how their work was distributed across the tasks of their project, to what extent that work was outside their job role, and what challenges they encountered in performing these tasks. With job roles on the y-axis and the tasks on the x-axis, each row of the matrix displays the proportion of people in the job role that perform a task (orange), as well as the proportion of those people who encounter gulfs of execution (green) and evaluation (blue) in performing that task.

Moreover, the on-diagonal boxes identify the tasks that correspond to the participants' job roles, as explained in Chapter 4.3. The red box bounds the "Developer" tasks (i.e., dataset management, debugging, image building, monitoring, and YAML/deployment logic) and are within the expertise of an infrastructure developer and library developer. The orange box bounds the "Operator" tasks (i.e., job submission and cluster observation) and are within the expertise of a cluster operator. The purple box bounds the "Scientist" task (i.e., AI/ML app source code) and

is within the expertise of a data scientist. The gray box bounds the “Other” task (i.e., application versioning), within the expertise of a project manager.

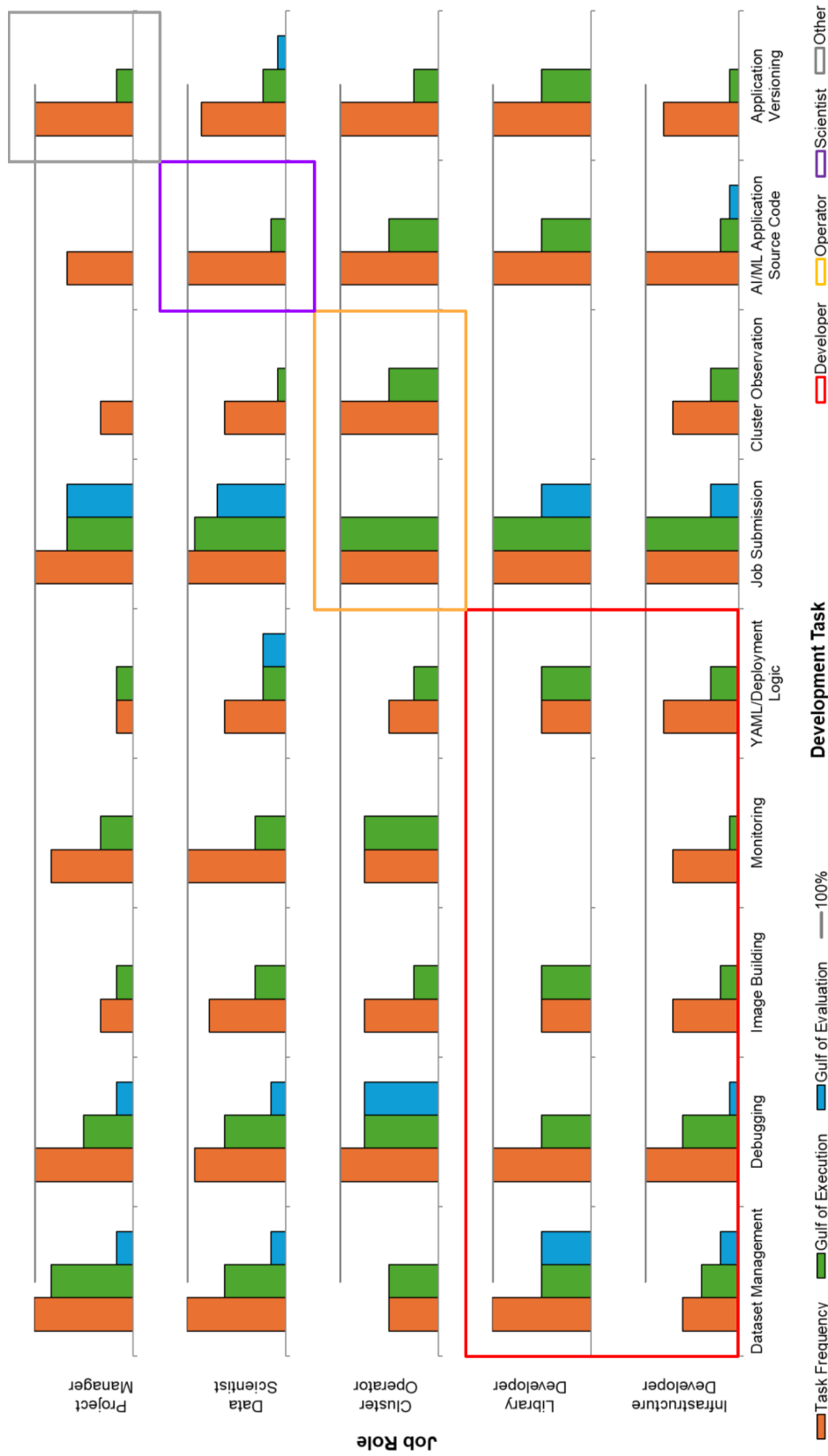


Figure 5.1. Combined tasks and gulfs of execution and evaluation, grouped by job role and development task for 35 participants, including: six project managers, 13 data scientists, four cluster operators, two library developers, and 10 infrastructure developers. Tasks and job roles are further categorized into their interpretations.

It is immediately apparent that, with few exceptions, everyone is doing a bit of everything, possibly requiring people to do tasks beyond their job role expertise or to spend time communicating with people who have such expertise. This reveals a high level of non-modular work. Notably, dataset management, debugging, and job submission tasks are not only commonly performed across most job roles as well as outside of our participants' defined expertise, but they also present significant challenges in terms of the gulfs of execution and evaluation. As stated in Chapter 4.1.2, a gulf of execution is defined as the difficulty of achieving the user's intentions with the available actions. A gulf of evaluation is defined as the amount of effort the user must exert to determine the physical state of the system and to determine how well expectations and intentions have been met.

Figure 5.1 builds on the general idea of how expertise mitigates the feelings of distance that a user experiences, as shown in Table 4.3. In reality, we find that our participants experience gulfs of execution and evaluation even when performing tasks within their expertise. This finding is consistent with Hutchins et al. [142], as although tasks that are sufficiently practiced often become, in a way, automated such that they require less cognitive effort, this does not eliminate the semantic distance that our participants experience given that the burden is on them alone, as the user, to bridge the gap, as stated in Chapter 4.1.2. Therefore, the complexity of performing a task outside of one's expertise does not disappear, rather, it shifts between two sides: 1) the system side and 2) the user side. Depending on the design of the system and the expertise of the user, the feelings of distance and engagement that a user experiences will vary and result in further support from either or both sides.

To apply the gulfs to the statements made by our participants, we defined them as the challenges that they reported experiencing and further categorized them into a challenge of either interpretation (evaluation) or taking action (execution). We further break down the challenges we categorized as gulfs to examine in depth to what extent they impact the feelings of distance and engagement that a developer experiences when performing a task within as well as outside of their expertise and familiarity. Table 5.1 further breaks down Figure 5.1 by providing an

overview of the specific challenges we found within the gulfs of execution and evaluation for dataset management, debugging, and job submission tasks.

Table 5.1. Summary of the specific challenges developers experience when performing dataset management, debugging, and job submission tasks for AI/ML containerized app development.

Task	Execution Challenges	Evaluation Challenges
Dataset Management	• Dealing with large datasets • Ensuring data quality through complex filtering processes • Continuous optimization of infrastructure and storage • Mounting storage and managing credentials using COS	• Adapting storage from HPC to Kubernetes • Overcoming the learning curve for COS • Infrastructure-application knowledge
Debugging	• Adapting debugging from interactive to batch sessions • Overcoming cluster unreliability with checkpointing and iterative debugging • Manual process of logging	• Multi-expert, real-time coordination • Inadequate tooling for log analysis • Limited visibility and uncertainty of an error’s root cause • Balancing observability and alert fatigue
Job Submission	• Computing resource contention and manual coordination • Users with varying experience and access • Switching among multiple tools due to a lack of integration • Infrastructure variability and hybrid cloud abstractions • Configuring jobs using YAML	• Managing unpredictable resource availability • Ineffective experiment tracking

In the following, we provide a detailed analysis of the extent to which employing Kubernetes for AI/ML containerized app development, in the context of these three tasks, affects our participants’ work and the emergence of these challenges.

5.2 Dataset Management

Dataset management tasks are within the bounds of the “Developer” job role and consist of curating and managing datasets, as defined in Table 4.9. Infrastructure developers and library developers perform dataset management tasks, as these are within the bounds of their job role expertise, as shown in the red box that surrounds all “Developer” tasks at the lower left of Figure 5.1. However, they along with participants from all job roles experience gulfs of execution and evaluation, with the exception of cluster operators who only experience gulfs of execution, when performing these tasks, as shown in the left-most column of Figure 5.1.

Regarding the participants whose job roles fall within the bounds of dataset management tasks, the overall percentage of library developers that experience execution gulfs also experience evaluation gulfs. In contrast, the overall percentage of infrastructure developers that experience

execution gulfs is much higher than those who also experience evaluation gulfs. In part, this may be due to a difference in the total number of participants we interviewed from each of these job roles, as we interviewed 13 infrastructure developers and only two library developers. However, we only conducted interviews with 35 participants and therefore, did not interview many people overall. Therefore, our findings regarding who experiences the most dataset management gulfs are a starting point for future work to consider.

It may also reflect the fact that infrastructure developers are more likely to possess a full-stack expertise to design, deploy, and optimize AI/ML containerized apps effectively, as they can evaluate all the components involved. Whereas library developers are less likely to possess this expertise and therefore struggle to evaluate whether inefficiencies stem from infrastructure misconfigurations or application inefficiencies and slows down progress, especially for AI/ML containerized app development.

5.2.1 Summary of Execution Gulfs

Overall, we found that participants experience distance when executing dataset management tasks for large-scale, AI/ML containerized apps, especially LLMs. The distance they experience is due to the demand for large, continuously growing datasets to be processed under tight deadlines. The distance they experience is further increased by the shared computing resources they use as well as the unfamiliar infrastructure and storage environments in which they execute dataset management tasks.

Most AI/ML containerized app development effort is spent on the dataset preprocessing stages, as ensuring the quality of a continuous collection of new, larger datasets adds repeated effort. This repeated effort widens the semantic distance that participants experience as dataset filtering processes scale, become more complex, and require further optimization. The semantic distance in the gulfs of execution that participants experience represents their need to constantly re-think their intentions. This re-thinking is due to their need to continually modify their job run configurations and source code to provide further support for their goals of scaling up or down

their jobs, depending on the shared computing resources available to them.

As dataset filtering processes scale, it increasingly takes more time to secure shared computing resources and takes longer for these larger jobs to run. This increased amount of time widens the articulatory distance that participants experience as they wait for large amounts of computing resources to become available. The articulatory distance in the gulfs of execution that participants experience represents the lack of readily available shared computing resources and the unpredictability of being able to secure them. This results in delayed feedback and interruptions that add to, rather than remove, the perception of an intermediary between participants and the cluster. Participants find such an intermediary to be unreliable as without a clear, continuous representation of the cluster's state, they cannot plan for when they will be able to secure computing resources. Although participants are able to use tools to actively monitor the cluster's state, other factors remain unclear. These unclear factors, such as job priority, impact the availability of shared resources and result in a lack of direct engagement.

These experiences of semantic and articulatory distance are further increased by the learning curve that participants must overcome when adapting to the new infrastructure and storage in Kubernetes environments from those used in traditional HPC environments. As well, participants' tool choices must shift with the growing scale of their datasets. This shift requires them to spend time learning how to use multiple tools. Participants switch between these tools as they iteratively scale dataset processing tasks. This switching increases their cognitive load and ultimately shifts the complexity rather than eliminates the burden of their task.

The following provides a thematic analysis of the gulfs participants encounter in executing dataset management tasks for AI/ML containerized app development, as listed the first row and first column of Table 5.1.

5.2.1.1 Dealing With Large Datasets

Participants use Kubernetes clusters to conduct multi-stage, iterative processing of large amounts of data for AI/ML containerized apps such as LLMs, spatial-temporal analysis, and

VR. These datasets range in size from terabytes to petabytes and are large enough to strain the computational resources that were available at the time the interviews were conducted. Data scientists emphasize that some stages of developing AI/ML containerized apps require more computing resources than others, such as data preprocessing, filtering, and tokenization. Therefore, these stages of developing AI/ML containerized apps widen the articulatory distance that they experience. For example, geospatial datasets are large (i.e., petabytes) and as a result, a data scientist must secure more computing resources as well as wait longer for preprocessing to finish compared to other stages of AI/ML containerized app development, such as model training:

(A23, Data Scientist) *“We’re talking about petabytes of data. It’s very hard to process, merge, filter, and apply other functions to this data. Sometimes it’s more resource hungry to pre-process this data than to train a model. I’m talking about a lot of hours just to do a simple function, using a lot of GPUs.”*

The frequent need to process large amounts of data on a limited amount of shared computing resources results in resource contention as GPU availability is disrupted by competing priorities. In high-performance computing (HPC) environments, job priority determines computing resource allocation based on Quality of Service (QoS) to ensure reliable performance as per Service Level Agreements (SLAs) between providers and users [178]. High-priority jobs receive more computing resources than low-priority ones to meet these commitments. Resource contention forces job pauses and disrupts progress as participants either wait for more computing resources to become available or reconfigure their job submission to use fewer resources on a subset of their data in order to continue making progress. The impact of these interruptions, which widen the articulatory distance that they experience, are exacerbated by critical deadlines for delivering products to clients. Project managers working with their teams on a preprocessing pipeline to conduct data annotations and filtering for large language model (LLM) applications deal with large datasets and as a result, job runs take additional time to complete.

A project manager said that AI/ML containerized app development requires a large amount of computing resources for a certain length of time, resulting in jobs getting preempted:

(A1, Project Manager) “It’s been very tight to do this all in one month because of the size of the data. (...) Sometimes we need to give the GPUs back because higher priority jobs come up. We need to stop our job and continue with a smaller amount of resources.”

Participants use various libraries to deal with smaller and larger amounts of data. For example, Although Pandas is good for preprocessing small datasets, using it with larger datasets makes for slow progress [58] and widens the articulatory distance that participants experience. To improve preprocessing speed and deal with the memory constraints of larger datasets, Dask and Ray are the two main libraries that participants use, both of which integrate with Pandas. Infrastructure developer and data scientist participants use tools such as Dask, Ray, and Pandas for executing dataset management tasks, as shown in Appendix B in Tables B.2 and B.3.

Dask is useful because it handles parallelism, broadcasting, and merging operations, which allows users to work with large datasets efficiently without having to manually manage these complexities [115]. This simplifies the process of scaling computations across distributed systems while maintaining a familiar Python style, which is the main language that participants use to write AI/ML app source code. Similar to Dask, Ray abstracts much of the complexity of distributed systems by allowing users to parallelize tasks with minimal code changes. This enables efficient processing of large datasets without requiring complex manual container orchestration (e.g., scheduling, fault tolerance, load balancing). It’s important for an infrastructure developer to find solutions that scale for more resource-intensive stages, like preprocessing, as these stages take up a majority of their time:

(B5, Cluster Operator) *“Dask and Ray, that’s the priority. For smaller datasets, you can use Pandas but once the dataset becomes larger, thousands of rows, Pandas is really slow and single threaded. In machine learning, preprocessing is at least 80 percent of the work.”*

Although cluster operators and data scientists may begin executing these tasks on a subset of data using tools such as Pandas, eventually they must switch to using other tools such as distributed libraries (i.e., Dask, Ray) to more efficiently scale preprocessing tasks. Exploiting these different levels of scale can save them time such that they can successfully accomplish their preprocessing tasks in fewer iterations, which means they don’t have to continually wait for their requested computing resources to be granted and for their preprocessing tasks to finish. However, employing a variety of tools to manage datasets results in additional time spent learning how to use multiple tools and this effort is compounded by the time it takes to switch between them. Therefore, this tradeoff reveals how their task complexity is shifted rather than absolved, as they must spend additional time learning how to use multiple tools and switch between them, increasing their cognitive load and widening the semantic and articulatory distance they experience.

5.2.1.2 Ensuring Data Quality Through Complex Filtering Processes

Data scientists comprehensively filter large datasets to ensure data quality before proceeding to subsequent stages in the AI/ML pipeline. Inadequate filtering can propagate errors and waste computing resources on flawed data and models. Filtering takes place to ensure data quality by removing harmful content such as hate, profanity, and abuse (HAP), personal information (PI), and non-compliant licenses. These stages are critical for ethical and legal compliance, however, they add significant overhead. Filtering the datasets is a large effort with dedicated teams of developers. Dealing with data licenses is outside of a data scientist’s expertise and control. In fact, dedicated teams of lawyers communicate such revocations or changes to them.

Data license revocations or changes force data scientists to repeat data filtering and ultimately, model retraining. This results in wasted computing resources and development time, as the semantic distance that a data scientist experiences widens from the additional effort of repeating their work:

(A7, Data Scientist) *“The lawyer would tell us, ‘No, this license we cannot use.’ You need to retrain the model, filtering this other license that they wouldn’t like. That’s really the three important things for the cleaning. HAP filtering, license filtering, and PI filtering.”*

Filtering is further complicated by multi-stakeholder decisions that add coordination overhead. Project managers and their teams must make careful decisions regarding the datasets used and consult with their clients on the factors that impact cost. Understanding the clients’ needs is important, as they often have different views than the developers themselves, who usually go with what their ultimate users, the clients, want. Communication is also needed with cluster operators, also known as cluster admins, to schedule time in advance for large, high-priority jobs on clusters that have the right requirements. Further communication is needed with cluster admins to scale up and down processes in order to be sure they are only using the computing resources that they need in order to preserve cost. Ultimately, although this communication is important, it is time-consuming as it adds additional steps that widen the semantic distance that a project manager and their team experiences, as well as contributes to the pressure they already feel in working under critical deadlines:

(A1, Project Manager) *“What thresholds should be chosen is a multi stakeholder decision. Because we are preparing data to facilitate model training, we involve the model training owners to sign off on what thresholds we are choosing. (...) We usually have an explicit decision about what to filter out.”*

5.2.1.3 Continuous Optimization of Infrastructure and Storage

Optimizing performance for large dataset processing is necessary to meet critical deadlines and improve turnaround times. Optimization efforts involve debugging job failures, addressing issues of slow job runs, and configuring clusters for increasingly larger datasets. Hardware and performance issues may interrupt or slow job runs and development progress, resulting in delayed feedback that widens the articulatory distance that participants experience. Large dataset processing is constrained by AI/ML pipeline stages that must happen sequentially and take considerable time, leaving little room for error as hardware failures, node crashes, or slow runs interrupt progress. For example, large datasets that are collected must go through complex processing before they are made available to model training teams.

While continuous data collection can happen in parallel to preprocessing, project managers emphasize that they and their teams must conduct frequent job re-execution with new, larger amounts of data, which increases the complexity of the job submission and widens the semantic distance that they experience. As a result, continuous optimization is needed to debug failures, speed up slow runs, and scale jobs with growing data demands in order to develop a competitive product as well as continually bridge the semantic and articulatory distance that a project manager and their team experiences:

(AI, Project Manager) “There is another team that’s continuously acquiring new data, so in the next month when we conduct the entire preprocessing pipeline over again, it’s not the same data. It’s new data. Every month we have been doubling the size of the data because the amount of data we need to have processed to be competitive with things like Llama 2 and ChatGPT is humongous.”

5.2.1.4 Mounting Storage and Managing Credentials Using COS

Data scientists unfamiliar with cloud object storage (COS) struggle with executing tasks such as mounting storage and managing credentials. In contrast to the file storage systems in a traditional HPC environment where data is organized as files in a hierarchical structure, object

storage is a technology that organizes data as objects with unique identifiers in a non-hierarchical structure [53]. Object storage is popular for cloud applications and has become increasingly popular for scientific applications with large datasets. COS is a subset of object storage that is hosted and managed in the cloud by service providers.

We found COS to be the most used tool for dataset management tasks, as shown in Figure 5.2. COS comes with performance and operational constraints, which are new challenges for data scientists accustomed to the file systems used in traditional HPC environments. We found that the main points of friction when executing dataset management tasks using COS come from the additional steps they need to perform, such as mounting storage as well as managing credentials and secrets. These additional steps increase a data scientist’s cognitive load and widen the semantic and articulatory distance that they experience:

(A24, Data Scientist) *“The biggest problem was managing the data across the different environments, for example, making it available locally. What people found easy with [organization A’s traditional HPC cluster] was that all the data was always in a file system where they could play around with it and do everything there. Mounting the COS repositories and all of that was probably the biggest challenge.”*

5.2.2 Summary of Evaluation Gulfs

Overall, we found that participants experience distance when evaluating dataset management tasks. The distance they experience is due to the additional steps they must perform when adapting their storage tools and processes from traditional HPC environments. The distance they experience is further increased by overcoming the learning curve for these new storage tools and processes. This learning curve requires participants to develop a more integrated understanding of infrastructure-application knowledge.

Additional steps such as caching and checkpointing create more decisions that participants must consider when performing dataset management tasks. These additional steps widen the semantic distance that participants experience as performance becomes a more prominent issue

in high-stakes, AI/ML containerized app development. The semantic distance in the gulfs of evaluation that participants experience represents these additional decisions that they must consider. These additional decisions come from the requirement to use an increased number of tools and processes in order to determine whether their goals have been achieved.

The articulatory distance in the gulfs of evaluation that participants experience represents the increased cognitive load of maintaining relevant information from multiple, non-integrated tools and processes. This increased cognitive load makes it more difficult for them to effectively monitor their progress and results in a lack of direct engagement. Switching between using multiple, non-integrated tools burdens their ability to quickly evaluate the outcome of their job performance and understand whether their dataset management is the root cause of any performance issues.

Participants must overcome the learning curve for using COS and gain the expertise to evaluate whether job inefficiencies stem from infrastructure misconfigurations or application inefficiencies. This learning curve widens the semantic and articulatory distance that participants experience. Participants with experience using traditional HPC environments that are unfamiliar with Kubernetes may assume that a storage system such as COS behaves like a traditional file system. However, COS introduces a different architecture that has its own performance constraints. Participants may be unfamiliar with or even unaware of these differences, which widens the semantic distance that they experience.

Without resolving their conceptual misunderstandings, participants underestimate job runtimes and inefficiently process data, slowing down their development progress. Without understanding how to evaluate such performance issues, participants frequently reach out for help from more experienced developers. Reaching out for help widens the articulatory distance that they experience as they spend additional time waiting for assistance.

These experiences of semantic and articulatory distance are increased by the limited resources available for education and support. These limited resources result in additional time spent stuck and reaching out for help from the few developers that have a more integrated

understanding of infrastructure-application knowledge.

The following provides a thematic analysis of the gulfs participants encounter in evaluating the outcomes of dataset management tasks for AI/ML containerized app development, as listed the first row and second column of Table 5.1.

5.2.2.1 Adapting Storage From HPC to Kubernetes

Participants with experience using a traditional HPC environment are faced with new challenges as they must evaluate how to adapt tasks such as training, fine-tuning, and inferencing to run in a Kubernetes environment. Unlike traditional HPC file systems (i.e., Ceph) where computation and storage are often tightly coupled for performance, COS completely decouples computation and storage [160]. Participants in all job roles, with the exception of cluster operators, use checkpoints for dataset management tasks, as shown in Appendix B in Tables B.2, B.3, B.5, and B.4. A library developer performing tasks such as training AI/ML models in a Kubernetes environment must make important decisions on evaluating caching and checkpointing:

(A2, Library Developer) “Think about distributed training and fine-tuning and inferencing in terms of Kubernetes and what it means to completely decouple compute and storage. There is no local storage. All the storage is an object store. (...) Normally people assume there is a network file system, which is attached. I can do '/home/something' and then lo and behold, everything is there. All those small assumptions make a huge difference in terms of how you think about your training jobs. What do you cache becomes a lot more important. If I had a network file system, I wouldn't think twice about what to cache, how to make checkpoints, because these models are a couple of tens of gigabytes in size or even a hundred gigabytes in size.”

Indeed, we found that the main points of friction when evaluating dataset management tasks using COS come from understanding performance tradeoffs as well as caching and check-

pointing decisions, which widen the semantic distance that they experience.

Data scientists unfamiliar with COS have misaligned expectations about its performance constraints, which can slow down their data processing tasks and make it harder for them to evaluate job run efficiency and completion times. For example, data must go through the network when using COS, which isn't as efficient as loading data from local storage. Misaligned expectations regarding COS lead to confusion, widen the semantic distance that data scientists experience, and frequently result in them seeking help from more experienced developers. This also results in them making slower progress on evaluating their dataset management tasks, which widens the articulatory distance that they experience. A more experienced data scientist said that novice data scientists must learn the tradeoffs of using COS and how to better optimize data processing tasks such as loading data in parallel with their data consumption:

(A8, Data Scientist) "There was a learning curve for [data scientists] to understand how COS works and the pros and cons of using it. For example, it can never beat local storage. If they don't know, they might come back saying, 'Why is the data from stack trace so slow?' (...) COS is one tool, job submission is another tool, and they need to learn how to merge the two together."

The above quote also regards how participants must use multiple, non-integrated tools. As said by the more experienced data scientist, novice data scientists must learn how to integrate these tools and processes themselves. This further widens the semantic distance that they experience. As well, switching between multiple tools further widens the articulatory distance that they experience. This results in a lack of direct engagement as a data scientist's cognitive load increases from the burden of having to maintain relevant information while switching between multiple tools.

Participants with experience using a traditional HPC environment that are unfamiliar with COS struggle to understand the important differences in a Kubernetes environment, which results in them spending time reaching out for help from more experienced developers. As a

result of how many data scientists struggle with mounting storage and managing credentials due to conceptual gaps, custom tutorials were developed by infrastructure developers to bridge the semantic distance that they experience. For example, a specific tutorial for how to create secrets. However, sharing these educational materials and making them widely available across many teams is a challenge that rests on a small number of infrastructure developers. This reveals that semantic and articulatory distance also exist due to limitations in the format and availability of these materials as well as those that create them. An infrastructure developer explains the knowledge gaps that data scientists with experience using a traditional HPC environment have when transitioning to a Kubernetes environment:

(A13, Infrastructure Developer) “[Data scientists] thought it’s like a file system, it’s not. (...) They [have knowledge] gaps. You are familiar with YAML, how to use KubeFlow. Now you get deeper, how you work properly with object storage on [organization A’s Kubernetes cluster]. For the critical workflow, we take care of it because we know it inside out. We help them, but we only have a few people. We need to share the information across more people, more teams.”

5.2.2.2 Overcoming the Learning Curve for COS

In general, participants experience a steep learning curve when employing new tools and concepts for dataset processing and management. Data scientists that transition to using an OpenShift Kubernetes platform from a traditional HPC platform need to learn new terminologies and storage configurations that widen the semantic distance they experience. An infrastructure developer said that transitioning data scientists to these new platforms requires demonstrating through benchmarking that their prior performance on HPC platforms can be matched on the new platforms:

(A13, Infrastructure Developer) *“I set up the cluster and help migrate the [data scientist] team to the OpenShift platform, help them benchmark, and ensure their performance is on par with the VM. Otherwise, they won’t move. We have a lot of AI folks new to OpenShift. We did a lot of education sessions, help them to use, for example, COS because it’s very different from how you do it in the traditional HPC cluster.”*

Indeed, many participants stated that learning how to use COS is the biggest challenge. Although some participants, such as the infrastructure developer from the above quote, create such educational materials and provide further documentation with examples for using COS, they are limited. For example, documentation provides commands that initially ease the process of getting started however, these commands contain many arguments that data scientists may be unfamiliar with. As a result, when it comes time to adapt these commands to run their own code on their own datasets, the semantic distance that they experience widens as they face these challenges, unaided by the documentation.

To help with the transition to using COS, participants suggest that the process itself could be more abstracted to simplify the learning curve as well as reduce the cognitive load and articulatory distance that they experience such that they do not need to spend as much time learning new terminologies and storage configurations. A library developer suggests that in addition to existing educational materials and documentation, a comprehensive course could be developed to explain how to use COS:

(A6, Library Developer) “There’s so much (...) to get to the next step, which is running code on your own dataset, which could be a large dataset with a variable number of GPUs. Because specifically, the dataset part, you need Cloud Object Storage in order to run things on [organization A’s Kubernetes cluster] with some large dataset. That in itself deserves a whole course because it’s not researcher-friendly at all. (...) Right now at [organization A], if you want to do this, then you have to find out what a Cloud Object Storage is, you have to install a lot of software where you have to figure out what a PVC is, you have to figure out what an AWS bucket is, you have to figure out how to get AWS credentials from your [organization A] Cloud Object Storage, which doesn’t make any sense. (...) Either make it simpler or make a course on how to do this.”

Choosing the storage system to use with a cluster can be a challenging task in itself, as there are many factors (e.g., capacity, performance, scalability, speed) to evaluate in providing participants with options to efficiently process large datasets [9]. Inexperienced participants are overwhelmed by these additional decisions they must make that widen the semantic distance they experience. In cases where participants are provided with various storage options, they need to understand the nuances among them to be able to choose the right one. Participants, such as the data scientist quoted below, can better understand how to select among various storage options by spending time reading any provided documentation or reaching out for help:

(B6, Data Scientist) “There are quite a few options that [organization B’s Kubernetes cluster] has. I was like, ‘Whoa, there are all these options. How do you even choose one?’ [There] is a lot [to consider].”

5.2.2.3 Infrastructure-Application Knowledge

Effective dataset management in AI/ML containerized app development requires tight integration between the application and infrastructure, which is difficult to achieve without a

full-stack understanding. In the context of developing AI/ML containerized apps on Kubernetes clusters, a full-stack understanding refers to comprehensive knowledge and proficiency across the entire technology stack, from the application layer (e.g., AI/ML models, frameworks, workflows) to the infrastructure layer (e.g., hardware, container orchestration, underlying systems) [197]. Such a holistic perspective provides participants with the expertise to design, deploy, and optimize AI/ML containerized apps effectively, as they can evaluate all the components involved. A lack of full-stack expertise makes it more challenging for participants such as library developers to evaluate whether inefficiencies stem from infrastructure misconfigurations or application inefficiencies:

(A2, Library Developer) *“The Kubernetes and the infrastructure underneath it, we’re sort of blind to it. We’re looking more at application, how to optimize your application on these run times for Ray, all the distributed platforms that were available. That was an acceptable mode of operation. But once you get into the AI, you need a tighter integration between the infra platform and the application.”*

The tighter integration between infrastructure and application in AI/ML containerized app development widens the semantic distance that library developers and other participants experience, and slows down progress, which widens the articulatory distance that they experience, especially for AI/ML containerized app development.

Due to this knowledge gap, participants frequently reach out for help with evaluating how they have managed their storage to be sure it is not the cause for their job run inefficiencies. Ultimately, we found that there are few of these full-stack experts that are able to support novice developers. This limitation is further discussed in Chapters 5.3 and 5.4, revealing how it creates a bottleneck that propagates beyond dataset management tasks to debugging, and job submission tasks as well. The propagation of this limitation to other development tasks shows how these tasks are part of a critical development cycle in which processes for task evaluation and knowledge sharing could be improved.

5.2.3 Dataset Management Tools

Figure 5.2 provides a list of the 31 tools used for dataset management tasks and illustrates that although one tool is used by most participants (i.e., COS), many other tools are also used. The “long tail” of tools indicates that participants require tools beyond COS and as such, must spend additional time learning how to use them all. Similarly, figure 5.4 demonstrates that across all job roles, participants use many, different tools for dataset management tasks. Figure 5.5 further demonstrates that across all job roles, participants use multiple tools for dataset management tasks. Therefore, the “long tail” of tools appears in all categories of job roles and as such, everyone bears the burden of learning how to use and switch between multiple tools. More tables that break down the tools that participants use for dataset management tasks by job role can be found in the Appendix B.

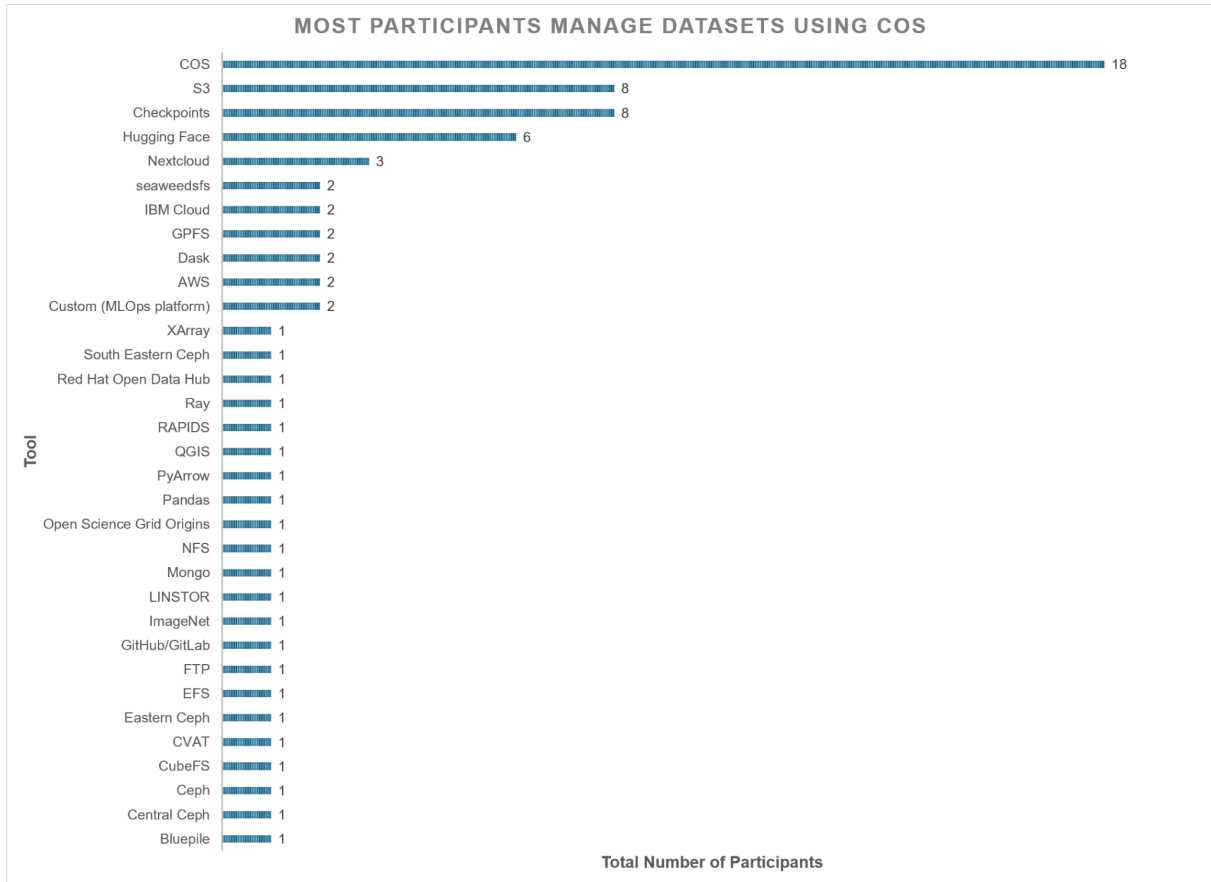


Figure 5.2. Overview of the tools used by participants for dataset management tasks. Numbers represent the total amount of participants that use each tool.

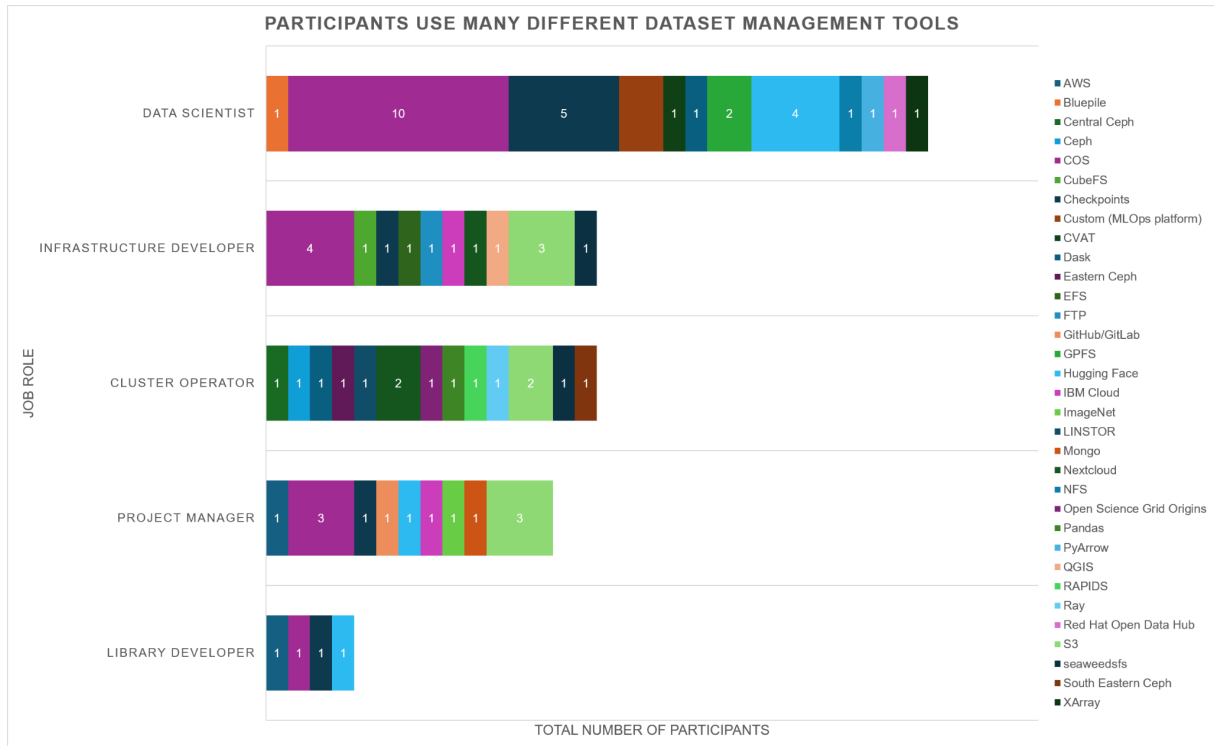


Figure 5.3. Breakdown of the tools used by participants with different job roles for dataset management tasks. Colors represent unique tools and numbers represent the total amount of participants that use each tool.

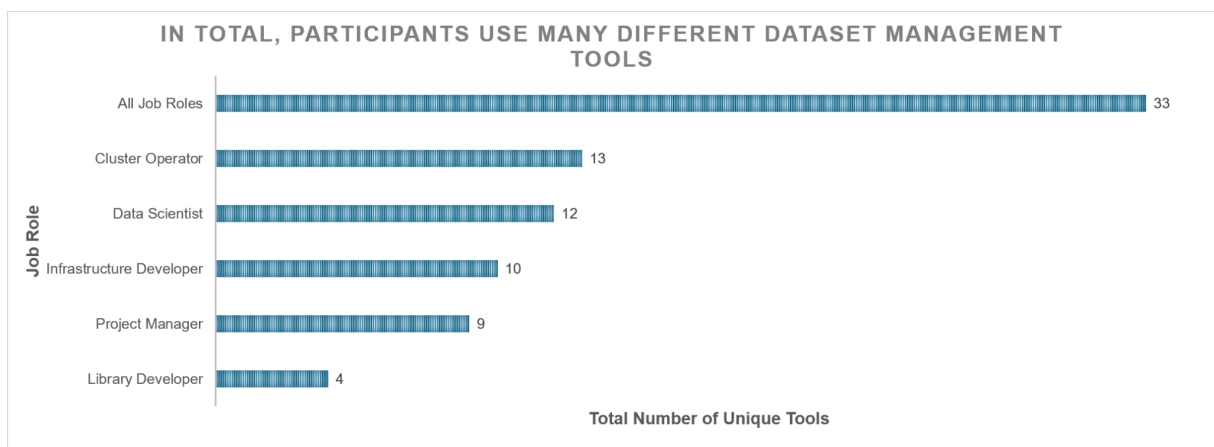


Figure 5.4. Overview of the total number of unique tools used by participants with different job roles for dataset management tasks.

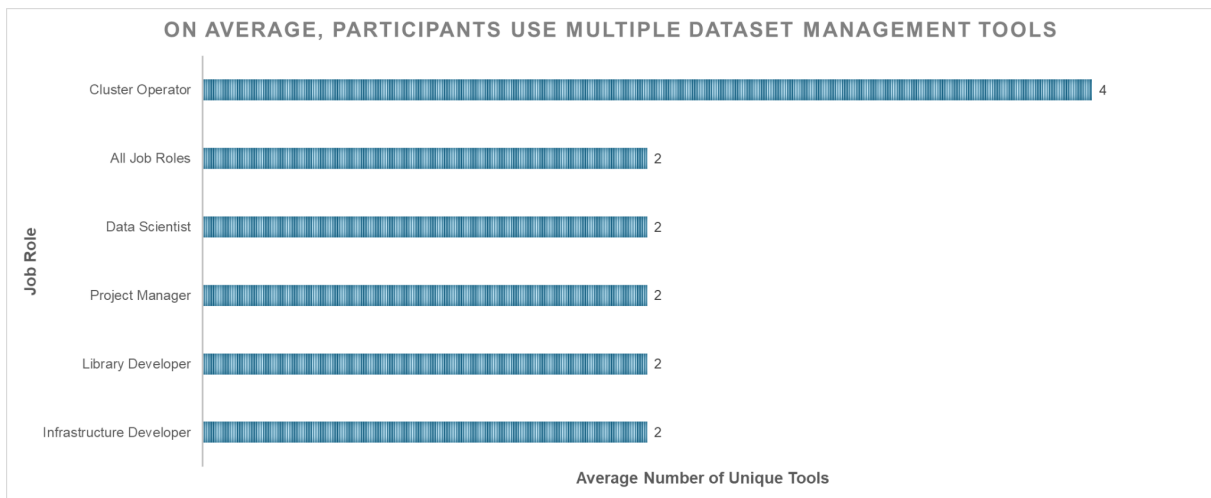


Figure 5.5. Overview of the average number of unique tools used by participants with different job roles for dataset management tasks.

5.3 Debugging

Debugging tasks are within the bounds of the “Developer” job role and consist of debugging an AI/ML job run to determine why its misbehaving, as shown in Table 4.9. Infrastructure developers and library developers perform debugging tasks, as these are within the bounds of their job role expertise, as shown in the red box that surrounds all “Developer” tasks at the lower left of Figure 5.1. However, they along with participants from all job roles, experience gulfs of execution and, with the exception of library developers, evaluation when performing these tasks, shown in the second left-most column of Figure 5.1. Therefore, nobody achieves modularity within their job role, as everyone must perform debugging. As discussed throughout this chapter, we find that participants with different job roles must depend on each other’s expertise to perform debugging in a combined effort, requiring frequent coordination.

Overall, the percentage of participants that experience execution gulfs is equal or higher compared to dataset management tasks. Furthermore, with the exception of library developers, the overall percentage of participants that experience evaluation gulfs is equal or higher compared to both dataset management tasks. In part, this may be due to the fact that debugging is a more general task than dataset management. It also may suggest that participants find debugging tasks to be more challenging than dataset management tasks. As discussed throughout this chapter, many participants find debugging tasks to be the most challenging of all.

Regarding the participants whose job roles fall within the bounds of debugging tasks, the overall percentage of infrastructure developers that experience execution gulfs is much higher than those who also experience evaluation gulfs. This may reflect the fact that infrastructure developers are more likely to possess a full-stack expertise to design, deploy, and optimize AI/ML containerized apps effectively, as they can evaluate all the components involved. No library developers experience evaluation gulfs. In part, this may be due to the fact that we interviewed only two library developers. However, we only conducted interviews with 35 participants and therefore, did not interview many people overall. Therefore, our findings regarding who

experiences the most debugging gulfs are a starting point for future work to consider.

5.3.1 Summary of Execution Gulfs

Overall, we found that participants experience distance when debugging large-scale, AI/ML containerized apps, such as LLMs and VR projects. The distance they experience is due to their goal of performing quick, interactive, low-friction debugging on Kubernetes production clusters, which are, in contrast, designed for non-interactive, batch workloads. These production clusters force slow iteration cycles in which participants must continually deploy and redeploy their job. Using these production clusters, they must perform additional steps such as extensive checkpointing, iterative testing, and manual logging. Furthermore, participants must find workarounds to resume progress when the cluster is busy.

The repeated effort involved in the slow, deploy-redeploy cycle, the additional, manual effort, and the workarounds widen the semantic distance that participants experience. This is especially true for those adapting their debugging processes from traditional HPC environments that offer interactive sessions. The semantic distance in the gulfs of execution that participants experience represents their repeated effort, the limited, automated support available from debugging tools, and the additional workarounds they must find to resume progress when interrupted by cluster delays. For example, participants may perform debugging in a local VM. In this case, they will switch from the Kubernetes production cluster to run their job in a different environment with less resources. These workarounds reduce Kubernetes' intended efficiency and widen the semantic distance that participants experience as they spend additional time switching between different cluster environments, reconfiguring their jobs in the process.

Participants under tight deadlines must spend additional effort in order to avoid issues of wasted time and computing resources. These issues are worsened by the slow iteration cycles in which participants use shared and unreliable computing resources. As discussed earlier regarding dataset management tasks, these issues also widen the articulatory distance that participants experience in debugging. They must either wait for computing resources to become available

or adapt their jobs to run on a smaller scale, to continue making progress. The articulatory distance in the gulfs of execution that participants experience represents the lack of readily available, shared computing resources. Such computing resource contention worsens issues of delayed feedback and interruptions from other tasks, such as dataset management. As well, the articulatory distance in the gulfs of execution that participants experience represents the lack of readily available, quick feedback from batch workloads. Batch workloads do not allow participants to change their job run configurations and source code as they are being executed, unlike the interactive workloads used in traditional HPC environments.

Ultimately, the shared and unreliable computing resources in Kubernetes production clusters, as well as the workarounds that ensue, make debugging the most time-consuming task of all, as discussed in Chapter 5.3.1.2. These feelings of semantic and articulatory distance are further increased by the pressure of the tight deadlines for delivering AI/ML containerized apps to clients, which also makes debugging the most painful task of all.

The following provides a thematic analysis of the gulfs participants encounter in executing debugging tasks for AI/ML containerized app development, as listed the second row and first column of Table 5.1.

5.3.1.1 Adapting Debugging From Interactive to Batch Sessions

Traditional HPC clusters support both interactive and batch sessions, with batch sessions being the dominant mode for workloads that run on production clusters due to their efficiency and scalability [52]. Development clusters and production clusters refer to distinct computing environments used for various stages of application development, testing, and deployment, particularly for AI/ML workloads [98]. A development cluster is designed for building, testing, and debugging applications during the development phase. It is typically a smaller, less critical cluster used by participants to experiment and refine code before deployment to production. A production cluster is a robust environment used to run fully developed, tested, and optimized applications in a live, operational setting. It is designed for reliability, scalability, and performance

to support critical tasks.

In the context of AI/ML containerized app development, although batch sessions are critical for executing large-scale AI/ML training jobs, interactive sessions support development and debugging. Interactive sessions are useful for prototyping AI/ML models, testing data pipelines with COS, or debugging GPU training scripts in real-time. Batch sessions are ideal for production AI/ML pipelines, where data is staged in COS, processed in containers, and the results are stored back in COS.

An interactive session is a real-time, user-driven session where a user directly interacts with allocated computing resources via a CLI or GUI. Interactive sessions allow for immediate execution and feedback for tasks such as debugging, testing, and exploratory analysis, which shortens the semantic and articulatory distance that participants experience in executing debugging tasks. For example, a data scientist starts an interactive session on a traditional HPC cluster, loads a dataset, and tests a PyTorch model on a GPU node, tweaking parameters in real-time. Resources are reserved through a job scheduler (e.g., SLURM, LSF) for the duration of the session and clusters often limit their duration or priority to prevent shared resources from being unavailable for too long. Waiting for jobs to launch on a traditional HPC cluster is faster with interactive sessions. A data scientist explains why they prefer interactive sessions for rapid iteration:

(A28, Data Scientist) *“That’s been how I’ve been doing it because it’s so much easier to get a GPU to try stuff on [organization A’s traditional HPC cluster]. And get an interactive session so you don’t have to wait for the job to launch every time.”*

A batch session refers to a non-interactive, automated job execution where a user submits a script or job description to a job scheduler which is run to completion on allocated computing resources, without user intervention. On a Kubernetes cluster, a user submits a YAML (e.g., specifying containers, resources, and commands) or a script and Kubernetes manages the container orchestration, scheduling, and execution of their job across the cluster. Batch sessions

allow for automated workflows for recurring, large-scale, production-ready tasks such as AI/ML model training. For example, a data scientist defines and executes a Kubernetes job via Kubeflow, which pulls the dataset from AWS S3, trains the PyTorch model across multiple GPU pods, and saves the model to S3, all without manual intervention. Batch sessions are ideal for automated, scalable, and fault-tolerant workflows, leveraging Kubernetes’ container orchestration for distributed AI/ML or data processing tasks.

However, despite these proposed benefits such as automation, that would shorten the semantic and articulatory distance that participants experience, instead, we found that they still experience them. Whereas interactive sessions exist on HPC clusters, they don’t on Kubernetes clusters. Although both traditional HPC and Kubernetes batch sessions are non-interactive, traditional HPC batch jobs rely on job schedulers optimized for tightly coupled, high-performance hardware, whereas Kubernetes batch jobs leverage container orchestration for cloud-native, distributed, or hybrid environments.

We found that participants, such as data scientists, experience points of friction from the lack of interactive sessions on Kubernetes clusters, which limits their ability to execute prototyping, test pipelines, tweak parameters, as well as more rapidly iterate and debug. These limitations widen the semantic distance that they experience. The lack of an interactive session forces data scientists to redeploy jobs for each debugging iteration on a Kubernetes cluster:

(A10, Data Scientist) *“The problem with working on [organization A’s Kubernetes cluster] is that, unlike [organization A’s traditional HPC cluster], [organization A’s Kubernetes cluster] is a production cluster. You are not meant to hold resources for longer than you are actively using them, which means that there’s no interactive sessions.”*

The lack of an interactive session is especially time-consuming when the cluster is busy and widens the articulatory distance that data scientists experience.

For participants that have experience using traditional HPC environments that are unfam-

miliar with Kubernetes, debugging requires additional time and effort to execute. For example, after updating their code, a data scientist must modify repositories, rebuild Docker images, and reapply YAML files, which widens the semantic distance that they experience. A data scientist explains how this contrasts with the simpler SSH debugging in a traditional HPC environment that requires a simple shell script to run:

(A9, Data Scientist) *“The reason I find [organization A’s traditional HPC cluster] to be more helpful or more easy is that it’s like a HPC environment. I can just SSH into it and debug stuff easily. But on the cloud, I have to make changes to the repo and rebuild the Docker image. Then, apply the YAML again.”*

Debugging jobs on a Kubernetes cluster involves significant overhead due to the need to repeatedly deploy and redeploy jobs, wait for resource allocation, and navigate cluster constraints. For example, a data scientist explains that they must wait for their job to finish to evaluate their progress and then re-execute their deployment after making adjustments:

(A10, Data Scientist) *“If you went to debug stuff, you have to keep deploying and redeploying and redeploying and redeploying and you watch the logs so everything gets set up, until something goes wrong. Then when something goes wrong, standard debugging process. Then, once you’re ready to try, you take down the job and you redeploy it.”*

Waiting for a job to finish can take even more time if the cluster is slow and widens the articulatory distance that data scientists experience. The iterative deploy-redeploy cycle in distributed environments ultimately slows debugging, increases turnaround time, and causes frustration.

5.3.1.2 Overcoming Cluster Unreliability With Checkpointing and Iterative Debugging

The low reliability of job runs necessitates extensive execution of testing, checkpointing, and manual checks to prevent costly failures. These processes complicate job submission

workflows and widen the semantic distance that participants, such as data scientists, experience. Checkpointing is critical to resume jobs without full, costly restarts and can be more challenging of a process than getting a training job itself to successfully run on a cluster. Checkpointing requires careful implementation to be able to resume a data scientist's jobs after a failure:

(A26, Data Scientist) *"[The reason why I can't run my job now] could be a higher priority job preempting the resources that were available. It's okay, I can restart something using the checkpoints, so when I restart my job, it takes off from where it stops, that's the reason for that."*

The need to execute rigorous testing and checkpointing adds overhead to job submissions, widening the semantic distance that participants experience, as they must anticipate and mitigate failures. Large-scale AI/ML jobs require error-free code and frequent checkpointing to avoid wasting computing resources after hours of running. Before submitting larger jobs that take up more computing resources, participants create jobs to execute on a smaller number of GPUs, using a subset of data. Large-scale AI/ML jobs amplify developer frustration, as failures can waste significant time and money. An infrastructure developer emphasizes the importance of executing iterative debugging steps to work up to running their jobs on a larger amount of resources:

(A12, Infrastructure Developer) *"Typically longer term jobs run on a larger number of GPUs because these are finalized jobs. There is more at stake when one of these jobs fail. For example, if you are running a job, on say 256 GPUs and it ran for three hours and then it failed, that's a lot of money."*

As stated by the infrastructure developer quoted above, if a larger job ends up failing after a few hours, that is costly training time wasted. Infrastructure developers and other participants emphasize using an iterative process for AI/ML containerized app development. Taking additional time to test and work up to running their job on a large amount of computing resources, along with checkpointing, will help prevent such wasted time.

Participants describe the debugging process as the longest of all the tasks they complete, with delays experienced when scheduling GPU resources and pulling large container images. These delays widen the articulatory distance that they experience. For example, when a participant runs the command `helm install`, this first creates a Python job. Then, a container is created with many, large container images. These images are located in the GPU, which takes time to bring up and, due to the unreliability of the cluster, sometimes the nodes are unavailable, which exacerbates progress delays. This issue prompts participants to execute workarounds that widen the semantic distance that they experience. A data scientist explains why they want to use local, VM debugging in order to simplify and stabilize their testing and debugging process:

(A24, Data Scientist) *“We are trying to come up with a mechanism where you can run locally in a VM for trial because getting a GPU, the resource, and scheduling it, and bring it to the point where you can debug is a long process. That is one challenge we generally run into.”*

Given that cluster reliability is low and the cluster environment is not always stable, participants using the cluster can only control how well-tested their own code is before submitting a large-scale job. Jupyter notebooks are used for executing piecemeal testing to avoid unclear failures when submitting large-scale jobs. Notebooks result in more direct engagement by allowing participants to easily see which parts of the code are running and where the issues are, as code is broken up to run in individual cells. Infrastructure developers and data scientist participants use Jupyter notebooks for debugging tasks, as shown in Appendix B in Tables B.8 and B.7.

Interestingly, infrastructure developers and data scientist participants experience the least amount of gulfs of evaluation for debugging, aside from cluster operators, as well as experience the least amount of combined gulfs of execution and evaluation for debugging overall, as shown in Figure 5.1. A project manager explains that by breaking up their code into cells, they can confidently test their code and get their work done faster:

(A1, Project Manager) *“Notebooks are very forgiving. If something goes wrong in one of your cells, then you can rerun it differently before submitting the whole job on 100 executors and seeing that everything failed.”*

5.3.1.3 Manual Process of Logging

As part of its scheduling framework, Kubernetes nodes run hundreds of objects with statuses and events [82, 85]. From these hundreds of objects, participants need to reduce their search space and focus on the objects that are important to their issue. After identifying the relevant objects, participants check the statuses and the events, which provide a brief amount of feedback. An infrastructure developer explains that in general, Kubernetes debugging involves manually navigating hundreds of objects, statuses, and events via `oc` or `kubectl` commands, which is time-consuming, lacks automation, and widens the semantic distance that they experience:

(A20, Infrastructure Developer) *“Once you say, ‘Ah, this is the event or the status of the resource that I’m really interested in.’ Then, you try to dig further inside logs for more context of what is happening. Everything now in this case is manual. You have to run bunch of `oc` or `kubectl` commands and then, figure out things.”*

The lack of automated tools to execute debugging as well as the complexity of distributed environments makes isolating root causes slow and computing resource-intensive. This causes delays to resolutions and widens the articulatory distance that participants experience. In general, root causes are difficult to determine, as discussed in Chapter 5.3.2. A data scientist explains how traditional debuggers are ineffective for distributed jobs and that they rely on manual methods like print statements to accomplish debugging, which widens the semantic distance that they experience:

(A9, Data Scientist) *“I use the typical VS code debugger, not anything on top of it. Most of these jobs are distributed. Debuggers don’t really help, you have to just use print statements or something. That’s the only good way of debugging these.”*

Inadequate or cumbersome logging and monitoring tools make it difficult to quickly identify and diagnose errors in large-scale workflows. Poor logging tools increase the time spent identifying errors, which widens the articulatory distance that participants experience. These limitations force participants to rely on manual execution processes or switch to using alternative cluster environments, which widens the semantic distance that participants experience. Data scientist participants use tools such as Aim for debugging tasks, as shown in Appendix B in Table B.7:

textit(A9, Data Scientist) *“[Aim]’s very helpful for logging long running experiments, so that once you go to sleep, you can check them out in the morning. Aim gives you a pretty nice view of what’s happening with the different statistics that you are logging. It’s pretty helpful in that sense.”*

As stated by the data scientist quoted above, Aim dashboards are used for analyzing and visualizing logs. The visualizations provided by Aim dashboards result in more direct engagement, as finding and fixing errors is an otherwise manual, time-consuming process.

5.3.2 Summary of Evaluation Gulfs

Overall, we found that participants experience distance when evaluating debugging tasks. The distance they experience is due to the difficulty of gaining clear, timely visibility into job failures. This difficulty is compounded by the complexity and unreliability of the Kubernetes environment that they use, which grants them limited user privileges.

Determining the cause of a job’s failure is a frequent, difficult task that requires participants with different job roles to depend on each other’s expertise and perform debugging in a

combined effort. Uncertainty regarding the cause of a job failure creates confusion when evaluating debugging tasks. This confusion widens the semantic distance that participants experience and forces frequent collaboration between multiple, specialized teams. Confusion comes from the non-deterministic behavior in Kubernetes environments, which widens the semantic distance that participants experience. Indeed, participants compare evaluating debugging tasks to a “black box” they cannot see into or a “dark basement” that they must blindly navigate. Furthermore, as our participants come from research organizations, they make use of novel hardware that introduces undocumented issues.

To resolve such issues, they must rely on sparse “tribal knowledge” among the few other developers they know who have worked on similar projects. The semantic distance in the gulfs of evaluation that participants experience represents their confusion regarding the root cause of a failure, which delays resolutions. This confusion is compounded by novel hardware and the limited number of resources participants have available to help them evaluate how to debug such failures. Although participants make use of multiple tools such as manual logging and alerts to monitor job progress, they are not well integrated. As well, they require trial-and-error to set up and achieve the right balance of relevant information. This adds repeated effort that widens the semantic distance that participants experience and delays the identification of the cause of a job failure.

The articulatory distance in the gulfs of evaluation that participants experience represents the increased cognitive load of maintaining all of the relevant information from these multiple, non-integrated tools. This makes it more difficult for participants to effectively evaluate their progress and results in a lack of direct engagement. Using multiple, non-integrated tools burdens their ability to quickly evaluate a job’s failure and understand what is the root cause. As well, the articulatory distance in the gulfs of evaluation that participants experience represents the increased number of people they must collaborate with in order to determine whether their goals have been achieved. Many participants with different job role expertise must come together in order to fully evaluate the cause of a job failure and cover the spectrum of possibilities, from the

infrastructure to the application. This severely limits independent work and slows high-stakes AI/ML containerized app development under tight deadlines.

These feelings of semantic and articulatory distance are increased by the limited user privileges in production clusters, which create barriers to understanding job failures and result in users with less privileges to reach out for help from those with more access.

The following provides a thematic analysis of the gulfs participants encounter in evaluating the outcomes of debugging tasks for AI/ML containerized app development, as listed the second row and second column of Table 5.1.

5.3.2.1 Multi-expert, Real-time Coordination

Job crashes are a frequent occurrence as the cluster can be unreliable. Debugging job crashes is an important, “high-priority” task that all participants deal with, no matter their job role expertise. Our interviews revealed that debugging jobs on Kubernetes clusters is inherently complex due to the involvement of multiple components that require varying job role expertise, which widens the semantic distance that participants experience. Debugging often requires coordinating multiple experts across teams in real-time calls to avoid prolonged back-and-forth when evaluating issues. This coordination is time-consuming and may be disruptive, as explained by a project manager and infrastructure developer:

(A1, Project Manager) *“We started this WebEx call in the morning because we have an issue and there is a Spark expert and there is a expert in tokenization. There is an expert in how the data was prepared. There is a cluster admin, so he can make changes in the cluster if we need to. When we have issues, which is almost every week if not every day, we need to get all of these people together.”*

(A12, Infrastructure Developer) *“Often times, a job crash is one of the more important, high-priority issues that we deal with, and that is one of the cases wherein whenever we have a problem we have to immediately interact with teams, different teams. There is an infrastructure team, there is a job dispatcher team. All of us sit down on calls and run through what the issue is.”*

As explained by the project manager and infrastructure developer quoted above, this coordination widens the articulatory distance that they experience, especially when problems arise weekly or daily.

In production clusters, limited cluster admin privileges force infrastructure developers to seek workarounds to evaluate issues and require them to communicate across multiple teams, which means communicating across different job role expertise, as explained in Chapter 4.3. These workarounds and additional coordination efforts widen the semantic and articulatory distance that they experience. As stated in the previous chapter on dataset management challenges, debugging performance issues in large clusters requires understanding both the underlying infrastructure and application behavior. An infrastructure developers explains why they develop custom tools to conduct automated health check tests for evaluating whether the source of a performance issue lies in the application or infrastructure layer:

(A13, Infrastructure Developer) *“Before, [debugging performance issues was] much easier for me because I have that privilege to do whatever I want. In production, not anymore, I have to find a lot of work around because we do see, especially the hardware issues. That’s something I have to develop based on my limited permission to try to be able to find out the issues, which node has issue. We have to build our own health check test. (...) Then, we tell the IT team to reboot or work with the cloud team to figure out issue.”*

Custom tools such as the one described in the above quote, help bridge the semantic distance that infrastructure developers and other participants experience from requiring such a full-stack understanding. More specifically, the semantic distance in the gulf of evaluating a debugging task is resolved by using custom tools to determine the root cause of an error, automating this task. However, as explained in the above quote, the articulatory distance is not bridged using such custom tools, as additional communication with other teams (i.e., the IT team regarding the infrastructure or the cloud team regarding the application) is still required. Note that, although in this chapter we are primarily discussing gulfs of evaluation, such custom tools also reduce the gulfs of execution, as determining the root cause of an error will help to employ the right team to address the issue.

These challenges described in the above quote are compounded by Kubernetes and OpenShift abstractions, which make it difficult to understand job statuses, performance impacts, or failure causes without full-stack expertise. Infrastructure complexity obscures job status, so participants struggle to understand how configurations affect performance or why jobs fail, which widens the semantic distance that they experience. For example, data scientists notice when jobs run slowly, however they lack insight into possible misconfigurations (e.g., suboptimal default settings, wrong network interfaces) and require expert intervention to help them evaluate their issues, as these debugging tasks diverge from their expertise. An infrastructure developer, referring to a data scientists seeking out their help, says that data scientist users only learn of

issues such as misconfigured defaults after their batch jobs complete and are found to have under-performed:

(A13, Infrastructure Developer) “What the user can know, can see, is that the job is running much slower. [Data scientists] usually come to us, ‘Why [is] my job [running] slower?’ Because you did not configure [your job] correctly. Because your default is not good (...) That’s why users have trouble to get the performance. (...) There’s a lot of learning for all of us, how to make a good default value.”

This only exacerbates the amount of time that a data scientist subsequently spends reaching out for help and widens the articulatory distance they experience.

Choosing a good default value can be challenging as it depends on various factors that participants must evaluate themselves. For example, default values depend on a participant's goal, the type of job they want to run and whether it needs to support a wide range of AI/ML pipeline stages such as training or dataset preprocessing. Another consideration is how a GPU should be leveraged for a job. For example, training jobs may prioritize performance. A further consideration is the network interface, as using the slowest one available will impact speed. All of these decisions widen the semantic distance that participants experience just in this one task of choosing a good default value.

Infrastructure developers often help others debug performance issues in large clusters as it requires understanding both application behavior and the underlying infrastructure, such as identifying PCIe or GPU memory issues, which is within their expertise. In a Kubernetes cluster, even though the nodes for GPUs are used by other workloads, there are ways to capture the desired information from the nodes in order to run benchmarks or custom health check test tools to identify the nodes that are the problem, and resolve the issues. For example, PCIe and GPU memory issues require a reboot to reset the cluster. Although some issues may not affect the performance of the application, others may, such as PCIe issues. However, infrastructure developers still experience gulfs of evaluation in debugging tasks, as shown in Figure 5.1:

(A13, Infrastructure Developer) *“You have to understand what [other teams] are doing and you also have to look at the infrastructure to find the root cause. It takes time, it takes experience. I don’t think it’s just because you don’t know application, because you are not the person daily run the job. You need to understand their pattern, how they use the GPU. It’s very challenging. It’s not just one thing.”*

Infrastructure developers, such as the one quoted above, still experience such gulfs partly because the challenge they face in helping others to debug is compounded by their need to understand the differences in how other teams submit their own jobs and what their goals are in using the cluster. Helping other teams involves communication across different job role expertise, as explained in Chapter 4.3.

A main concern of participants is to be sure that they are effectively using all of the resources that they request. Participants use Grafana dashboards to monitor their own jobs and avoid overloading GPUs or memory, which can block GPU usage for others. Grafana gives participants insight into the many factors that must be considered when running a large-scale job on shared resources within a single dashboard, which shortens the semantic and articulatory distance that they experience. For example, GPU usage, how GPUs are loaded, and the amount of GPU memory and cores used throughout the duration of their job run. Participants use Grafana to visualize communication among CPU, GPU, and the network, as others also need to download and upload data in a shared cluster environment. Infrastructure developer and data scientist participants use Grafana for debugging tasks, as shown in Appendix B in Tables B.8 and B.7.

Tools such as Grafana, along with custom tools that perform automated health check tests, speed up the debugging process by allowing an infrastructure developer to work more independently, rather than having to spend time reaching out to other developers for help with evaluating their job and result in more direct engagement:

(A13, Infrastructure Developer) “We expose those information to Grafana. Then, we can go to the dashboard and figure out which nodes have issue. Then, we don’t have to manually do what we did in the past six months. (...) We can go to the dashboard, find out the nodes that had issue, and fix it. Instead of every time go to [redacted] to run the test. That makes my life easier.”

Participants face intermittent node failures which are beyond their control and require debugging by specialized teams, complicating progress evaluation and widening the articulatory distance that they experience. Grafana dashboards help participants, such as data scientists, understand job terminations due to resource constraints by visualizing metrics for their running jobs. However, as explained by a data scientist, job terminations are only discovered after checking the dashboard, rather than when they execute their job submission:

(A26, Data Scientist) “That’s interesting, maybe my job got killed [helm list]. I can check that here if I go to a Grafana dashboard [switches to browser to Grafana dashboard], it tells me how many jobs am I running [hovers over “Current Assign...”]. I guess I’m not running anymore, looks like my job got killed [scrolling through list of “GPUs per Namespace”]. These are all the jobs that are running at a time [”GPUs per Namespace”].”

Job terminations, such as the one faced by the data scientist quoted above, add uncertainty to job scheduling and require participants to switch between multiple tools, increasing the semantic and articulatory distance that they experience.

Participants struggle to monitor and evaluate job performance due to inadequate tools or abstractions that hide critical details. Infrastructure developers invent job scheduling tools to handle Kubernetes scheduling, create containers, and allocate the resources for the containers to run. These details are abstracted away from data scientists that use the cluster to conduct fine-tuning of their AI/ML models. Infrastructure developer, project manager, and data scientist

participants use this custom job scheduling tool for job submission tasks, as shown in Appendix B in Tables B.13, B.15, B.12.

On one hand, the abstractions provided by tools such as the custom job scheduler and Helm templates simplify job submission for novice participants and reduce their cognitive load, shortening the semantic and articulatory distance that they experience. On the other hand, the low-level issues are obscured (e.g., tainted GPUs, network connectivity), which limits participants' ability to evaluate performance and widens these feelings of distance. As a result, participants that conduct various stages of AI/ML containerized app development (e.g., training, fine-tuning) must work together to produce a model despite experiencing different challenges and using different terminology. An infrastructure developer explains why this makes it more difficult to evaluate related issues and help one another:

(A12, Infrastructure Developer) "This is where abstraction I think plays a big role. While I'm aware of the broad architecture of say [organization A's Kubernetes cluster], I do not dabble on a day-to-day basis on, are my Ethernet cables properly connected or which GPU is probably tainted, things like that. For me, I can run this Helm job, and then I can look at once it is deployed and start doing my job once again. Similarly, for downstream people who train, for example, me and a few other people in the team, we are very much involved in the weeds. We are dealing with what's the running data, what's the batch size? What is the gradient? But if you go to the people who are doing fine-tuning, they have different kinds of challenges. Even the terminology is different. We all are involved in building the model, that's the commonality."

5.3.2.2 Inadequate Tooling for Log Analysis

Current tools for accessing and analyzing logs are insufficient, verbose, or crash-prone, complicating the evaluation of debugging task progress. Ineffective log analysis tools make it difficult to quickly assess debugging progress, as it is time-intensive to sift through irrelevant

or excessive logs, which slows down error identification and progress evaluation. These tool limitations widen the semantic and articulatory distance that participants experience. Participants, such as infrastructure developers, explain that looking at the cluster logs is one of the biggest challenges and is where they spend most of their time. As a result, they seek workarounds to enable local, VM debugging in order to speed up the process of getting their job running and quickly test it. However, these workarounds widen the semantic distance that they experience. An infrastructure emphasizes that they could benefit from more integrated tools (e.g., VS Code plugins) that filter relevant logs without overwhelming systems or crashing, as verbose logs hinder efficient debugging:

(A20, Infrastructure Developer) “VS Code is another open-source IDE. Someday, if there are nice tools that integrate in bringing at least logs that are relevant to me and then, you’re not crashing my laptop when it becomes verbose. Getting those would be really helpful for my day job. Some plugins and then, you hover here and then you see logs, like magic.”

As explained by the infrastructure developer quoted above, the lack of integrated tools to streamline log access forces manual inspection, slowing down the identification of critical errors and widening the articulatory distance that infrastructure developers and other participants experience.

5.3.2.3 Limited Visibility and Uncertainty of an Error’s Root Cause

Participants have limited visibility into the cluster’s state and infrastructure components, which hinders progress evaluation for debugging tasks. For example, network issues are difficult to debug due to lack of access to the underlying systems. These limitations widen the articulatory distance that participants experience. Determining the origin of network issues can require a project manager and their team to spend additional time coordinating with other developers across multiple teams, with varying job role expertise:

(A18, Project Manager) *“Where does the problem start? This is what we are seeing, our TCP retransmits are through the roof and it’s very intermittent. That’s all we see, I have no visibility to the switches to the host OS, I have no visibility to any of that. The debugging process now is, I stepped away from that call, about thirty people on one Webex trying to figure out what could be the problem.”*

Developers explain that their normal, everyday experience of debugging cluster issues is filled with frustration and uncertainty. Even the most experienced project managers struggle with debugging cluster issues. A project manager compares evaluating debugging tasks to a “dark basement” that they must blindly navigate, which reveals the extent of the semantic distance that they experience:

(A18, Project Manager) *“The way I’ve described it is, I’ve been in a dark basement for a long time. I know where all the bumps, where all the low beams are. (...) Every once in a while, something new comes out. You just have to figure out where the new object is.”*

A Kubernetes environment introduces non-deterministic behaviors and redundant error messages, making debugging painful and unclear, even for the most experienced users. We found that even these experienced users experience semantic distance in debugging a job running in a Kubernetes environment. Interestingly, we found that many of the participants we interviewed said they were not comfortable using Kubernetes. A cluster operator explains that despite how much experience they have with using Kubernetes, they still do not feel comfortable using it:

(A17, Cluster Operator) *“I’m never comfortable with Kubernetes, I pretty much don’t like it. I have an aversion against the concept, which is non deterministic. When you bring up a Kubernetes cluster, it’s throwing everything into a mix, there are no dependencies which are being solved in a regular way, you wait until the containers settle with each other. (...) In order to debug Kubernetes, it’s painful, you need really deep experience of it and it’s all a black box.”*

Implementing new hardware (e.g., H100 GPUs) to advance the cluster introduces novel issues that are difficult to evaluate due to limited prior knowledge and documentation. These novel issues widen the semantic distance that participants, such as project managers overseeing the development of the Kubernetes cluster, experience. Furthermore, solutions to such novel issues may only spread as “tribal knowledge” as developers must reach out for help from the few, if any, others they know that are using similar Kubernetes clusters. A project manager explains that these unfamiliar hardware issues slow debugging for them and their team, as they must rely on trial-and-error or involve multiple other teams, which further delays resolutions:

(A18, Project Manager) *“Not anybody else has H-100, so you try to figure out what it could be. I found issue in PyTorch that could have, which reminds me I have to update that issue so the next person that runs into this. It happened to be a weird GPU driver issue that was causing a CUDA issue that surfaced as trying to access a GPU that doesn’t exist. That’s normal.”*

Despite all of the well-designed tools available to use off-the-shelf and the custom tools invented by developers themselves, there is no simple solution when it comes to evaluating the issues of a Kubernetes cluster. A project manager explains that they and their team are constantly dealing with new issues when evaluating debugging tasks, which widens the semantic distance that they experience:

(A18, Project Manager) *“My partner, his line is, ‘Every day is a new problem never seen before, no idea how I’m gonna get through it.’ And by four PM is like, ‘Took care of that, that’s done.’ We’re constantly dealing with new odd things that not many people have seen. It’s the exact same problem on [organization A’s Kubernetes cluster]. There’s no easy tool to say, ‘This is how you fix this problem.’”*

5.3.2.4 Balancing Observability and Alert Fatigue

Maintaining effective observability through dashboards and alerts is difficult, as too many redundant alerts flood the system and cause fatigue, although too few miss critical issues, complicating progress tracking. Poorly balanced observability tools obscure debugging progress, as teams struggle to prioritize relevant issues or miss them entirely. Participants must achieve the right balance of relevant information, which adds repeated effort that widens the semantic distance that they experience as well as delays the identification of the cause of a job failure. As well, the increased cognitive load of maintaining all of the relevant information from multiple, non-integrated tools, widens the articulatory distance that participants experience, which makes it more difficult to effectively evaluate their progress as well as results in a lack of direct engagement. Participants make use of tools such as Matrix, an open-source and self-hosted alternative to Slack, for sharing alerts to team channels in a more integrated manner. Cluster operators, data scientists, infrastructure developers, and project manager participants use Matrix for evaluating debugging tasks, as shown in Appendix B in Tables B.8 and B.10. A cluster operator explains how they refine Matrix alerts to balance capturing important issues without overwhelming themselves, as excessive alerts reduce attention to critical problems:

(B3, Cluster Operator) *“This [redacted] team is now working on cleaning up and making sure that we only get important stuff in our Matrix. If we get too many, we stop paying attention. If we don’t get enough, then we don’t catch the important stuff. It’s important to keep this balance of what alerts we’re getting in Matrix from there.”*

Dashboards such as Grafana pull data on node usage and resource allocation, however, participants still need to understand how to manually evaluate whether issues such as stuck pods or storage configurations are the root cause of an error, which adds uncertainty to job scheduling and progress evaluation.

5.3.3 Debugging Tools

Figure 5.6 provides a list of the 34 tools used for debugging tasks and illustrates that although one tool is used by most participants (i.e., Slack, logs), many other tools are also used. These tools include both off-the-shelf and custom tools, invented by the participants themselves and indicated by the word “Custom” preceding the name of the tool in parenthesis. Communication tools are indicated by the word “Communication” preceding the name of the tool in parenthesis, or more generally using the format of communication (i.e., in a meeting or otherwise in a 1-1 interaction).

Interestingly, many participants indicate that they reach out for help debugging, revealing a lack of modularity in their development process. These tools for communication indicate how independent work becomes impossible and as such, communication becomes necessary. Similarly, figure 5.8 demonstrates that across all job roles, participants use many, different tools for debugging tasks. Figure 5.9 further demonstrates that across all job roles, participants use multiple tools for debugging tasks. Therefore, everyone requires additional time not only to learn how to use and switch between multiple tools, but to reach out for help. For example, manual logs only partially support participants’ progress before they must reach out for help. More tables that break down the tools that participants use for debugging tasks by job role can be found in the Appendix B.

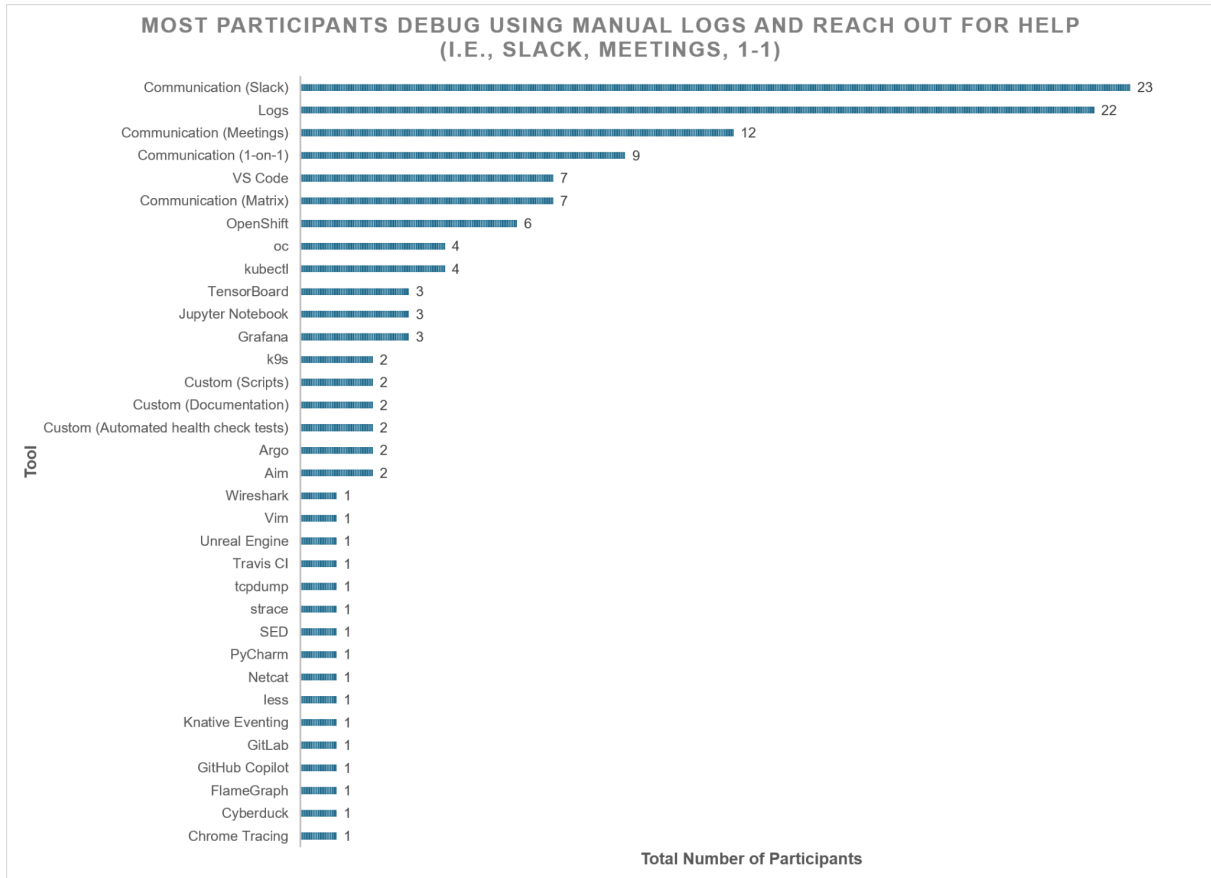


Figure 5.6. Overview of the tools used by participants for debugging tasks. Numbers represent the total amount of participants that use each tool.

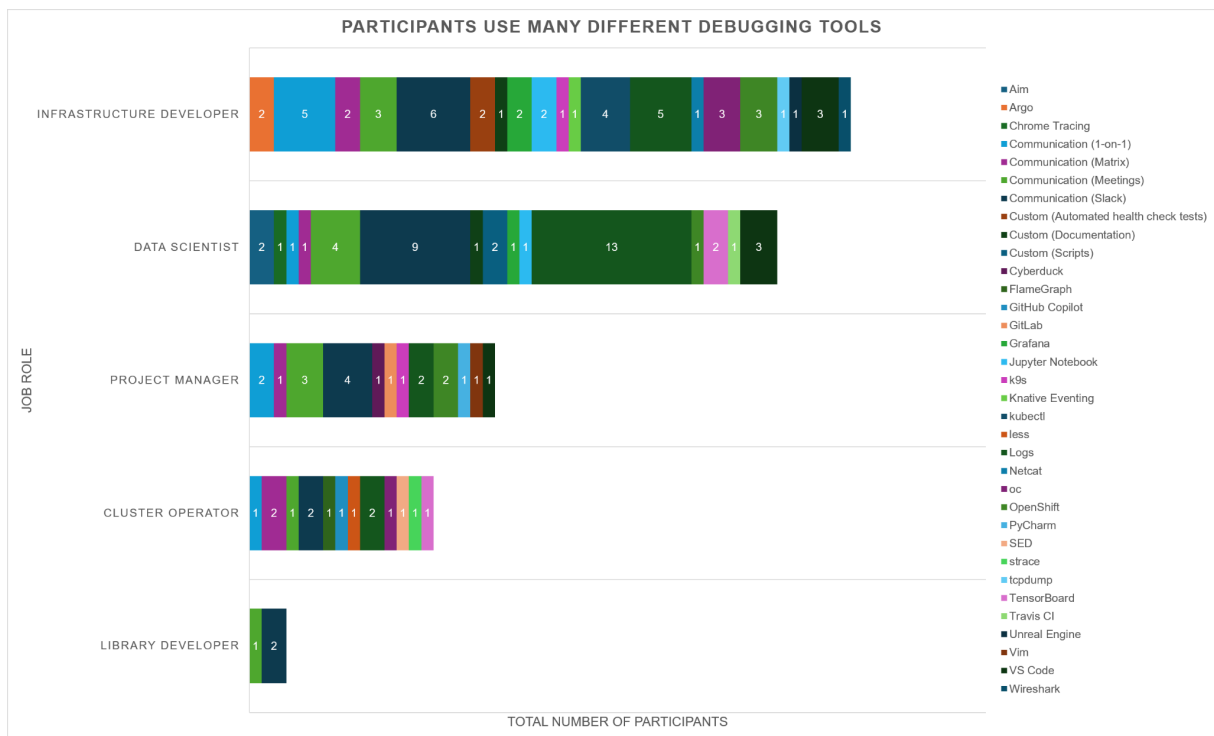


Figure 5.7. Breakdown of the tools used by participants with different job roles for debugging tasks. Colors represent unique tools and numbers represent the total amount of participants that use each tool.

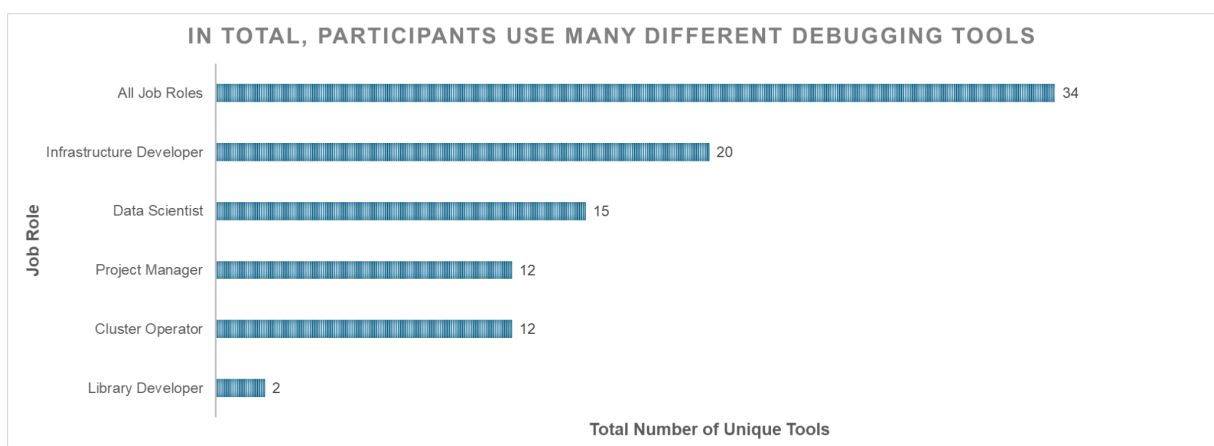


Figure 5.8. Overview of the total number of unique tools used by participants with different job roles for debugging tasks.

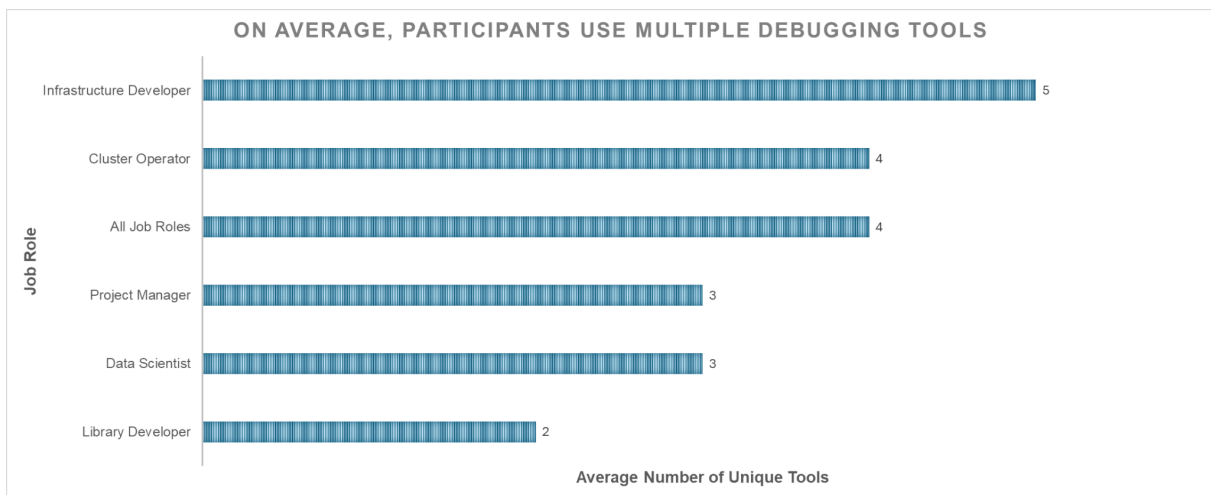


Figure 5.9. Overview of the average number of unique tools used by participants with different job roles for debugging tasks.

5.4 Job Submission

Job submission tasks are within the bounds of the “Operator” job role and consist of debugging an AI/ML run to determine why its misbehaving, as shown in Table 4.9. Cluster operators perform job submission tasks, as these are within the bounds of their job role expertise, as shown in the orange box that surrounds all “Operator” tasks in the middle-right of Figure 5.1. However, they along with participants from all job roles, still experience gulfs of execution at a rate of almost 100%, as shown in fourth right-most column of Figure 5.1. Moreover, participants from all job roles, with the exception of cluster operators, experience gulfs of evaluation. In part, this may be due to the fact that we interviewed only four cluster operators. Therefore, similar to debugging tasks, nobody achieves modularity within their job role, as everyone must perform job submission tasks. As discussed throughout this chapter, we find that participants with different job roles must depend on each other’s expertise to perform job submission tasks in a combined effort, requiring frequent coordination.

Regarding the participants whose job roles fall outside of the bounds of job submission tasks, the overall percentage that experience execution gulfs is equal or higher compared to both dataset management and debugging tasks. Moreover, with the exception of library developers, the overall percentage that experience evaluation gulfs is higher compared to both dataset management and debugging tasks. In part, this may be due to the fact that we interviewed only two library developers. However, we only conducted interviews with 35 participants and therefore, did not interview many people overall. Therefore, our findings regarding who experiences the most job submission gulfs are a starting point for future work to consider.

5.4.1 Summary of Execution Gulfs

Overall, we found that participants experience distance when executing job submission tasks for large-scale, AI/ML containerized apps, such as LLMs and VR projects. The distance they experience is due to the shift from the simple and direct process of running jobs in a

traditional HPC environment to the more complicated and indirect process of running jobs in a Kubernetes environment, which requires using additional job configuration tools and processes with limited user privileges.

The additional job configuration tools and processes that participants use are especially time-consuming when developing AI/ML models, which must be iteratively tuned. These additional tools and processes adds more steps that widen the semantic distance that participants experience as their cognitive load increases along with the risk for introducing errors. The friction introduced by these additional steps is amplified by computing resource contention, for which participants must find workarounds in order to resume progress when the cluster is busy. These workarounds involve finding other available computing resources, such as local VMs or other traditional HPC environments. As a result, participants spend additional effort to reconfigure their jobs to run elsewhere. The semantic distance in the gulfs of execution that participants experience represents the additional tools and processes that developers themselves must integrate in order to continue making progress.

High computing resource demands from competing AI/ML jobs creates computing resource scarcity and disrupts independent workflows. As discussed earlier regarding dataset management and debugging tasks, these issues also widen the articulatory distance that participants experience in submitting jobs, as those with low-priority jobs face long wait times to secure resources, experience job preemptions, and are forced to scale down in order to continue making progress. Although participants continue to make progress when they are forced to scale down their job, it further delays the final product, especially when developing large-scale AI/ML models. The articulatory distance in the gulfs of execution that participants experience represents the lack of readily available, shared computing resources. Such computing resource contention further worsens issues of delayed feedback and interruptions from other tasks, such as dataset management and debugging.

In general, further negotiation with cluster admins is required to secure large, high-priority computing resource allocations and this additional communication is time-consuming.

Some participants can't negotiate independently and need bosses or others to do so on their behalf, which adds overhead that widens the articulatory distance that participants experience. Therefore, the articulatory distance in the gulfs of execution that participants experience also represents the increased number of people they must involve in order to achieve their goals. More experienced participants with different job role expertise must come together in order to help novices submit a job. Novice data scientists struggle with making the decisions that come with the additional steps of employing Kubernetes for job submission tasks, such as computing resource estimation, configuration, and provisioning. As a result, novices, such as data scientists, rely on infrastructure developers and cluster admins for help.

These feelings of semantic and articulatory distance are increased by the limited user privileges in production clusters, which create a hierarchy where only the cluster admins are allowed to make low-level changes, the namespace admins may run large-scale jobs, and leaves the basic users to depend on these two higher tiers. This severely limits independent work and slows high-stakes AI/ML containerized app development under tight deadlines. These execution gulfs illuminate the gaps between ideal workflows and real-world constraints in high-stakes, resource-limited settings for AI/ML containerized app development.

The following provides a thematic analysis of the gulfs participants encounter in executing job submission tasks for AI/ML containerized app development, as listed the third row and first column of Table 5.1.

5.4.1.1 Computing Resource Contention and Manual Coordination

Securing sufficient computing resources is difficult due to high demand, competing priorities, and limited availability, which impact participants' productivity in executing job submission tasks. Resource scarcity and contention delay job submissions by requiring manual coordination and prioritization, which disrupts participants' independent workflows and widens the articulatory distance that they experience. GPU contention exists, in part, due to high demand from the AI/ML model training jobs or the AI/ML production-ready models that constantly need

to be run. An infrastructure explains that high-priority jobs such as data preprocessing secure resources quickly, however, low-priority jobs face delays or preemption:

(A12, Infrastructure Developer) “While we have namespaces and stuff wherein each namespace has its own set of resource quotas, oftentimes you would have priorities like we really need the data processing to be done. For example, we are told, ‘Reduce your job submission so that you can free up some GPUs so that others can use it because they have a higher priority that needs to be scheduled.’ So it’s a lot of collaborative effort among different teams.”

Coordination is required in advance to secure computing resources, even for high-priority jobs, and widens the articulatory distance that participants experience. If coordination is not timely enough, participants face delays in their work and risk missing critical deadlines for delivering their products to clients. High-priority jobs are more computing resource-intensive and costly to run. An infrastructure developer explains that in order to secure large amounts of computing resources on a shared cluster, participants must spend time communicating with cluster admins and have some established trust that they will use these resources efficiently:

(B4, Infrastructure Developer) “There’s a few groups that are leveraging all of the machines in [organization B’s Kubernetes cluster]. A much smaller cohort of researchers. That generally involves a closer relationship with the cluster admins, communication. You can’t just arbitrarily say, ‘I’m gonna use half the cluster to train my LLM today.’ It requires some communication and planning.”

Participants frequently spend time coordinating to secure computing resources, which involves negotiating with one another for priority. However, not all participants are able to negotiate, as some, such as data scientists, require another person on their team to do so on their behalf (e.g., bosses), adding overhead to communications for securing computing resources and further widening the articulatory distance that they experience. Rather than use the word

“negotiate” some participants describe the situation of resource contention as more dire, sharing how participants are “fighting” for the computing resources they require to make progress in their work. A data scientist explains that high-priority jobs may preempt others, forcing them to restart their jobs with reduced computing resource requests or wait, as is commonly seen in negotiations for GPUs:

(A16, Data Scientist) “There is a lot of coordination. Cluster admins don’t feel comfortable killing more jobs to give me 192 GPUs, so they ask me, beg me, ‘Can you do with 152 GPUs?’ I say, ‘Okay.’ They even ask if I can do with less and I say, ‘No, give me 152.’”

Balancing the availability of computing resources in a Kubernetes cluster with the demand for using them to run various kinds of jobs with different priorities is a difficult task. There are times where high-priority jobs take up most of the cluster’s computing resources. As a result, other participants, such as data scientists, must make due with the limited computing resources that are left, which slows their progress and widens the articulatory distance they experience. For example, a data scientist explains how a limited resources restrict them to executing one experiment at a time:

(A26, Data Scientist) “It’s a matter of resources, as there are more resources, we’ll get more people to use [organization A’s Kubernetes cluster]. But if you can only run experiment at a time, then, it’s too many people on [organization A’s Kubernetes cluster]. It’s really a matter of where the resources are. It’s tough to get GPUs these days.”

Computing resource contention is a problem that could be better handled. Participants suggest that maintaining multiple computing resource platforms (e.g., traditional HPC cluster, development and production Kubernetes clusters, VMs, bare-metal infrastructure) to run jobs is important to mitigate progress delays in times of resource contention. Furthermore, maintaining

familiar traditional HPC platforms while participants learn how to use a new Kubernetes platform can help shorten the articulatory distance they face from resource contention. However, there are tradeoffs in maintaining multiple different cluster environments beyond the added cost. For example, switching between these environments widens the semantic distance that participants experience as they must adapt their jobs to run in a different environment. A bare-metal infrastructure could be provided, which involves running an OpenShift Kubernetes platform directly on physical servers to leverage the hardware’s full capabilities without the overhead of virtualization [83, 61]. A library developer explains that having a portion of the cluster’s computing resources exist in a bare-metal infrastructure could allow for more experiments, unconcerned with containerization, to be executed in parallel:

(A2, Library Developer) *“It is a very, very good plan to carve out [organization A’s Kubernetes cluster] or any compute infrastructure into two parts. One is this OpenShift based infrastructure and one is this bare-metal infrastructure. Because when somebody wants to experiment with new technology, you don’t want all the containerization, the virtualization, all those overheads, it’s just another source of trouble for that.”*

As explained by the library developer quoted above, providing a bare-metal infrastructure for experimental jobs would help to avoid the OpenShift Kubernetes platform’s virtualization overhead as well as shorten the semantic and articulatory distance that they and other participants experience.

The intricate process of submitting jobs on a Kubernetes cluster is time-consuming and error-prone compared to on a traditional HPC cluster. The frustration that participants experience in initially requesting computing resources to run their jobs on a Kubernetes cluster differs from their prior experience using a traditional HPC cluster, and widens the articulatory distance they experience. A data scientist explains that in contrast, a traditional HPC cluster has no such overhead and provides them with a direct line of access via Slack to cluster admins:

(A23, Data Scientist) *“I use Grafana to monitor my own jobs internally, not the entire cluster. [Organization A’s traditional HPC cluster] is different because I think how people are using it, it’s kind of this omnipresent entity that processes things for us. People go there and submit a job and that’s it. If we have any issues, we put on Slack and we tag the [organization A’s traditional HPC cluster] admins.”*

Participants also contend with the newfound pressure of using shared computing resources with careful regard for not overloading the cluster and reserving more resources than necessary. A data scientist describes how it feels like “everyone’s watching” them use the cluster when they submit a large-scale data preprocessing job:

(A16, Data Scientist) *“That was a surreal experience for me. The first time, the journey was bumpy. I got to use 152 GPUs and Grafana is showing them being used, everybody’s watching.”*

The pressure of using shared computing resources is, in part, due to having to follow cluster policies. These cluster policies are often unspoken “gentlemen’s agreements” and thus widen the semantic distance that participants experience. Data scientists explain that not abiding by these policies widens the articulatory distance for other developers using the cluster, as they face further delays from the lack of available computing resources when the cluster is overloaded:

(A23, Data Scientist) *“I think the most important part is this unofficial policy, we could say, ‘gentlemen’s agreement’ on usage of the cluster. We are sharing the cluster with a lot of other people. So if you misuse it and submit a job in a way that will take all the memory or all the GPUs in a node, but then you leave GPUs hanging. Then, other people cannot use those GPUs because you’re using all the GPUs. This hidden agreement, that’s the hardest thing to pass.”*

Balancing speed, monetary cost, and priority in executing job submission tasks leads to tradeoffs that are not ideal. When faced with critical deadlines to deliver products to clients,

speed is often prioritized over cost. This can lead to maintaining large worker pools to avoid scale up and down delays, despite the high costs. A project manager explains that although this shortens the articulatory distance that they and their team experience from such delays by decreasing their dependency on other teams to scale up and down the resources they are using, this tradeoff is “not ideal” regarding the cost:

(AI, Project Manager) “When we have very urgent timelines, we don’t have time to scale down, scale up. We keep it running and this is something that I have to go and make sure that the executives are okay with because of the cost. (...) We create a massive worker pool, keep it on, and we run when we need to. This has really helped remove the scale up, scale down times and dependency on other teams and all of these things. But it shouldn’t be this way. It’s really not ideal.”

As explained by the project manager quoted above, they must communicate with other teams, specifically cluster operators, in order to scale up and down their jobs. From a usability perspective, one of the goals of using a Kubernetes cluster is to be able to automatically scale up and down jobs as resources become more or less available. Therefore, ideally, a Kubernetes cluster should shorten the articulatory distance that its users experience. However, in practice, more caution is used in a shared cluster environment with regards to deployment. As a result, deployment is done manually by users with admin privileges, such as cluster operators, as discussed in Chapter 5.4.1.2.

Project managers and their teams design tools that help AI/ML containerized app developers independently experiment without needing access to costly computing resources or to communicate with a data scientist. Custom tools for ML model experimentation offer foundation models as a service via multiple modalities (e.g., APIs, SDKs, GUIs) to simplify access to large models, similar to OpenAI [56]. Participants are offered multiple tool modalities as they have different layers of abstraction to make it easier to get started, get the model to do what they want, and plug it into the application they are building. A project manager overseeing the

development of such a custom tool explains that without them, participants needed to find an already, pre-trained API for a model that could accomplish their specific goal, or they needed a data scientist and a large amount of labeled data to create a model themselves, which act as significant barriers that widen the semantic and articulatory distance they experience:

(A19, Project Manager) “The difference in flow is like maybe they wouldn’t have tried a larger model because they didn’t have time or the resource or the access to do so, and now they can plug one in and see how it works. I had an application developer on my team say, ‘If you told me a year ago, I was going to be building something with AI models.’ Before, they would have needed a data scientist to help them, but with foundation models, you can often just engineer a prompt, talk to it in a different way and it’ll do the thing.”

5.4.1.2 Users With Varying Experience and Access

Participants emphasize that the more cumbersome parts of their process for executing job submission tasks could be automated, especially for novice developers, such as data scientists. After initially obtaining access to the cluster itself, data scientists repeatedly execute manual tasks each time they plan to submit a job, including identifying the right cluster to run a job on based on the computing resources they need, configuring and scheduling the job, and if the job fails, restarting the job. These repeated, manual steps increase the semantic distance that they experience. Infrastructure developers provide data scientists with help to estimate their job’s computing resource needs because they struggle with this and often guess, as configuring a job to run on a Kubernetes cluster is outside the bounds of their job role expertise. This additional help widens the articulatory distance that they experience. As well, it is a signal for the semantic distance that they experience due to not understanding how to independently make these decisions. A project manager acknowledges that the current state of their job submission process requires data scientists to have expertise beyond their training and widens the semantic and articulatory distance that they experience:

(A1, Project Manager) *“Also remember that you are asking for people to be, at the same time, Spark experts, Kubernetes experts, and data scientists. We have very few people like that and we’re not going to have more, we’re going to have less going forward. You want to take that out of their hands and capture as much manual processes as possible.”*

Job submission on a Kubernetes cluster requires manual intervention from cluster admins for executing provisioning, scaling, and debugging, as other participants may lack the sufficient permissions to independently accomplish these tasks, despite their expertise. These participants rely on cluster admins to create namespaces and add workers. Depending on the availability of computing resources, executing cluster provisioning itself can take time and results in delays from the required back-and-forth communication throughout the process of submitting a job, which widens the articulatory distance they experience. Once resources are available, the job can finally run, however additional communication is required in order to scale processes up and down so that participants are not inducing higher costs by taking up resources they are not using, which further widens the articulatory distance they experience. A project manager explains that the dependency on manual cluster admin processes slows the job submission process and can increase costs, which highlights the need for automated cluster provisioning:

(A1, Project Manager) *“We don’t have permissions to add workers to worker pools or isolate the workers for our job so that they’re not interfering. Every time we are starting something, we need someone from the research IT team who has all those permissions, ‘Okay, let me create your namespace, let me add you to that, let me add workers, oh, we don’t have capacity now, too bad, let’s wait.’”*

From another usability perspective, infrastructure developers express the difficulty they face in their inability to deploy updates to staging and production clusters without cluster admin assistance. Restricted permissions delay their progress in setting up and improving the cluster

itself for others to use. Deployment is done manually by a few cluster admins and can be infrequent, which indicates a low bus factor such that many depend on those few cluster admins to resume progress. As a result, infrastructure developers experience frustratingly long wait times for cluster allocations, which widens the articulatory distance they experience:

(A4, Infrastructure Developer) “What I’m trying to implement, something that is at infrastructure level, to get those things deployed on either staging or production, you have to wait and hope that you will get an allocation, because all the deployment is done manually by a few people, and that doesn’t scale, that doesn’t really work.”

As stated by the infrastructure developer quoted above, if they need to redeploy automated health check test tools or log in to get into the pod to run something, they are not able to independently do so.

Infrastructure developers acknowledge the caution taken by cluster admins as a reason for why deployment is done manually by a few people. However, infrastructure developers that work on implementing changes for the improvement of the cluster infrastructure explain that they need to be able to manage more tasks such as deployment themselves:

(A4, Infrastructure Developer) “I get that being an admin and deploying stuff on [organization A’s Kubernetes cluster] is not something you do easily. You need to be careful with what you do. But if I’m implementing things for the infrastructure, I need to be able to manage those things myself. The deployment process on staging and production is something that, in my opinion, absolutely doesn’t work.”

Infrastructure developers, such as the one quoted above, recommend that this manual deployment process be improved in the future such that they are able to more independently manage the tasks required of their job role expertise, shortening the articulatory distance they experience.

Participants emphasize that varying levels of users exist in a Kubernetes platform, which

require different permissions to execute their tasks. This is similar to how participants' job roles require specific training and expertise. Some participants work on the cluster, such as the cluster admins and infrastructure developers, while others use the cluster, such as library developers and data scientists, and yet others, such as project managers, fall in between these two categories. For example, an infrastructure explains how namespace admins have more control over large-scale jobs, whereas basic users are restricted to simpler tasks such as spawning Jupyter notebooks attached to GPUs to run their jobs, creating a dependency on higher-tier users that widens the articulatory distance they experience:

(B4, Infrastructure Developer) “There’s something to be teased out about a basic user versus advanced user that I would probably call a namespace admin. It’s sort of a logical separation that I’ve observed where we have the cluster admins, which are the people that can actually go and modify the machines themselves and update the Kubernetes version and device drivers, those kind of low-level technical aspects of the cluster. Then, there’s the sort of namespace admin tier, which are usually a senior person at a lab, the PI of a research group, or just an advanced [organization B’s Kubernetes cluster] user that’s been around a little while. These are people that have permission to run large-scale jobs on the cluster, provision lots of resources, a bunch of CPUs and a bunch of GPUs for a big job, or set up some specific network configuration. Then, under them in the tree, there’s a big swath of simple users that interact with what the namespace admin has set up for them. I see these three tiers of user-dom within the cluster.”

To expand on the above quote, [organization B’s Kubernetes cluster] provides a public portal to manage the namespaces that exist on the cluster. Each namespace is listed by name and includes detailed information. For example, the PI of the research namespace, the grants that are funding the PI’s research, and the PI’s institution are listed. The namespace admins that are responsible for the software running in the namespace environment are listed. The users

that are the students and others working on the project who may or may not be involved with the DevOps aspects of running the namespace are listed. There is a built-in hierarchy of users, separated by user and namespace admin. Compared to namespace admins, these users make up most of the the [organization B's Kubernetes cluster] population and perform “entry-level” tasks such as spawning Jupyter notebooks attached to GPUs (e.g., for training an AI/ML model). This indicates a low bus factor such that many depend on the few who have higher user privileges.

5.4.1.3 Switching Among Multiple Tools due to a Lack of Integration

Some participants prefer to use IDEs such as PyCharm over CLIs such as Vim because the IDEs alleviate the cognitive load of remembering commands, APIs, and functions through hints. Therefore, such IDEs allow for more direct engagement. Furthermore, participants express the desire to have a more integrated development process where they don't need to switch among multiple tools to execute job submission tasks, as they believe this could make their workflow more efficient. Switching between tools widens the semantic and articulatory distance that participants experience as they must understand how to use multiple tools and figure out how to integrate them into their development process such that they can use the output from one into the other more readily. IDEs such as PyCharm offer more integration (e.g., Git check-ins), however, some participants prefer command-line control, indicating a lack of seamless tooling and doubt over whether they are using tools to their full potential. A project manager explains that the effort to integrate tools is a “maddening” task left to the users themselves:

(B2, Project Manager) “[The tools] work well together because we make them work well together and which for the most part is maddening.”

The lack of robust automation and integrated tools increases manual effort, slowing job submission execution and complicating progress. Tools such as a custom job submission CLI simplify job submission execution and alleviate participants' cognitive load by walking them through the steps of submitting a job. As a result, the semantic and articulatory distance that

participants experience is shortened. Participants in all job roles with the exception of cluster operators use this custom tool for job submission tasks, as shown in Appendix B in Tables B.13, B.12, B.15, and B.14. The following quote is broken up into several pieces, with the first piece covering the first four decisions that a data scientist must make when using a custom job submission CLI to assist them:

(A26, Data Scientist) *“I use [a custom job submission CLI]. (...) The way I invoke it is like this [on the terminal]:*

```
$ ml/torchx/run/submit -y -- --data_path /data
--param_path /data
```

And then, it says how many workers [Number of workers: 6]. It’s not gonna submit it because I’m limited by resources, but I can just pretend to start one of those. So let’s say six workers [Choice 1: Configure your Resource Requirements ... Number of workers: 6], 64 cores, CPU cores per worker [CPUs per worker: 64], eight GPUs per worker [GPUs per worker: 8], memory [Memory per worker: 300], [Ephemeral Storage per worker: 5Gi]. Now storage, and this is that path [Choice 2: Code and Base Image ... Path to source: /Users/[redacted]/speech-fm/C-T2.20] that I have on my Mac to the model training part to the, PyTorch folder, so I can keep the same one. All right, Kubernetes context [Choice 3: Choose a Kubernetes context ... speech-fm/[redacted]/[redacted@[redacted].[redacted].com = prior choice]. Right, what was I thinking? It should say ‘speech-fm’ [Choice 4: Target Kubernetes Namespace ... Create a namespace], maybe it doesn’t, maybe have to renew my credentials. For that I do, I use ‘oc’ for that [switches to the browser to login.redacted.redacted.com], [Log in with id], [Display token], I have to run this command [switches back to the terminal to enter the token to log in]:

```
$ oc login -token=sha[redacted] --server=[redacted]
$ ml/torchx/run/submit
```

The above quote by the data scientist reveals how custom job submission tools are not used in isolation, as they must switch between different tools to access further information, such as their login credentials. The quote is continued below to further illustrate how such custom job submission tools are used and covers 10 subsequent decisions that a data scientist must make when using a custom job submission CLI to assist them:

(A26, Data Scientist) “See now it says a namespace ‘speech-fm’ [Choice 4: Target Kubernetes Namespace ... speech-fm] which I want. This is an image that contains libraries that I need for my application [Choice 5: Select an image pull secret ... all-icr-io], built on top of PyTorch. All the data that I access is stored on COS AWS S3 [Choice 6: How do you wish to access S3 data? ... My credentials are stored AWS-style = prior choice]. Default profile [Choice 7: Choose an AWS Profile ... default = prior choice]. Now in my code, I assume that those COS buckets are mounted locally [Choice 8: How does your code access S3 data? ... My code assumes it will be pre-mounted as a local file system = prior choice]. For example, here I’m gonna access both, this one [Choice 9: Choose an S3 Bucket (note space selects and enter accepts) ... en90k; payload20k]. And I’m gonna use the performance storage [Choice 10: Choose an S3FS Storage Class ... imbc-s3fs-cos-perf = prior choice]. I’m gonna mount the first bucket as ‘/data’ and the second bucket as ‘/data2’ [Choice 11: Where should the S3 bucket be mounted in the filesystem? ... en90k: /data; payload20k: /data2]. I guess that’s [Choice 12: Choose a Pod Scheduler ... User the Multi-user Enhanced Kubernetes Scheduler = prior choice]. Okay [Choice 13: Choose a Pod Scheduler ... My administrator has already installed and configured = prior choice]. I’m gonna run with default priority [Choice 14: Job priority ... Default priority = prior choice] and now it’s submitting the whole thing and I think it’s gonna submit it, but it’s not gonna run because I’m at capacity.

The above quote by the data scientist reveals how custom job submission tools relieve

some of the cognitive load involved in making the many decisions required in order to submit their job, as they do not need to recall all of these steps themselves and are instead being prompted by the CLI along with being provided with options they have previously used. The quote is continued below to further illustrate how such custom job submission tools are not used in isolation, as a data scientist may run into an error when submitting their job, requiring them to switch between multiple tools in order to debug the error:

(A26, Data Scientist) “If I do ‘kubectl get’ or I guess I can do ‘helm list’:

```
$ helm list
> NAME
> redacted-torchx-98imu9v73gta
> redacted-torchx-tnlyqtucicuw
```

I have two jobs (...), this one is running [redacted-torchx-98imu9v73gta], and this one is scheduled [redacted-torchx-tnlyqtucicuw], but it’s not gonna be able to run because I don’t have resources. But we’ll get pods:

```
$ kubectl get pods
> No resources found in speech-fm namespace
```

Ah, that’s interesting. Oh, maybe my job got killed. I can check that here if I go to (...) a Grafana dashboard [switches to browser to Grafana dashboard]. It tells me how many jobs am I running [hovers over ‘Current Assign...’]. I guess I’m not running anymore, looks like my job got killed [scrolling through list of ‘GPUs per Namespace’], these are all the jobs that are running at a time.

Switching among various cluster environments (e.g., a traditional HPC cluster and a Kubernetes cluster) can be challenging as it increases cognitive load and the potential to introduce errors, widening the semantic distance that participants experience. There is no unified tool or workflow to easily run jobs across various cluster environments, which requires manual

adjustments to accomplish. Code must be edited to work on specific clusters (e.g., adding and removing environment-specific flags, storage paths, or arguments), which commonly leads to errors if changes are not properly managed during commits. The need to remember and apply environment-specific changes disrupts productivity and introduces friction in development and deployment processes. Comparing logs or outputs across various clusters is difficult due to the lack of integration, complicating debugging and analysis. A data scientist explains that these challenges highlight the need for better tooling integration to enable consistent workflows across diverse cluster environments, particularly for AI/ML containerized app development tasks:

(A28, Data Scientist) “In terms of these different environments, I would say they’re not integrated. In a perfect world, I would have my code on my laptop and then if I want to run it on one GPU, I would use some tool to launch it and say, I want one GPU. If I want to launch a bigger training on [organization A’s Kubernetes cluster], I would use the same tool and just say, but please do it on [organization A’s Kubernetes cluster]. (...) In reality, it’s not really like that, I have to edit my code on [organization A’s traditional HPC cluster].”

Switching between using a traditional HPC cluster and a Kubernetes cluster for submitting AI/ML jobs requires implementing different approaches, with Kubernetes relying on complex YAML files to define resources, although traditional HPC clusters use scripts. Writing large, intricate YAML files for Kubernetes (e.g., defining resources, storage, and job parameters) is time-consuming and error-prone, especially compared to simpler traditional HPC scripting. Environment-specific configurations such as data paths differ among clusters (e.g., COS such as AWS S3 for Kubernetes, local file systems such as Lustre for traditional HPC) and require manual adjustments for data access and job parameters. No single tool supports both clusters, which forces participants to adapt workflows for each environment’s unique requirements, widening the semantic distance that they experience. To address these challenges, participants create Bash scripts to abstract and unify job submission across both clusters. For example, a single YAML

file specifies parameters such as dataset paths (e.g., COS for Kubernetes, local for HPC), AI/ML job parameters (e.g., learning rate), and reduces the number of manual environment-specific tweaks a participant needs to make. Bash scripts generate Kubernetes YAML files automatically and submit jobs to both clusters, minimizing manual YAML writing, ensuring consistency, and reducing errors. Data scientist participants use custom scripts for job submission tasks, as shown in Appendix B in Table B.12. This approach saves time, simplifies workflows, and enables seamless switching between HPC and Kubernetes clusters for AI/ML workloads:

(A7, Data Scientist) “You could have all sort of different parameters, but what I wanted to have was just one simple YAML file that could be used with a script on [organization A’s traditional HPC cluster], but with another script on [organization A’s Kubernetes cluster] as well. This is really what I’m trying to do because writing YAML for [organization A’s Kubernetes cluster], is extremely complicated. Let’s say I did it once and I have the script, so I interact all with a Bash script on [organization A’s Kubernetes cluster]. ”

5.4.1.4 Infrastructure Variability and Hybrid Cloud Abstractions

Differences in cluster configurations, network performance, and hardware (e.g., organization A’s Kubernetes cluster versus client environments) require tailored job submission execution strategies. Although it is useful for participants to have multiple cluster environment options to continue making progress in times of computing resource contention or as they are learning how to use a new cluster environment, there is a tradeoff. Infrastructure variability forces participants to adapt their job submissions to specific environments, which increases complexity and reduces portability, as well as widens the semantic distance they experience. For example, a data scientist explains that training jobs optimized for organization A’s Kubernetes cluster’s A100 GPUs may fail on clients’ older GPUs (e.g., V100, P100), requiring adjustments, and network bottlenecks that are specific to an organization’s clusters necessitate custom solutions for job performance:

(A8, Data Scientist) *“Whatever you can successfully run in [organization A’s] cloud might not be the same story if you want to run in a client’s environment. Those are things we also need to deal with.”*

From a usability perspective, one of the goals of developing a Kubernetes hybrid cloud platform is to be able to better address differences in cluster configurations and issues of resource contention to avoid disruptions and the complicated reconfigurations required to continue making progress. Therefore, ideally, a Kubernetes hybrid cloud platform would shorten the semantic distance that its users experience. For example, an infrastructure developer explains that migrating jobs among clusters requires minimal changes in an OpenShift Kubernetes platform, as users only need to make small, independent changes to their own image and YAML files:

(A13, Infrastructure Developer) *“For the hybrid cloud platform that we’re pushing, it does make a user’s life easier when you move from one cluster to the other. Now we are doing move from the TCP kind of network into RoCE. There are things you need to change in general, like your image and in the YAML file, a few lines of change. In general, you can keep everything the same, you use your same software stack, whatever you have in your container, because the rest, the operator is already there, no change. You only really need to worry about your piece.”*

However, as mentioned in Chapter 5.3.1, for participants with experience using a traditional HPC environment that are unfamiliar with Kubernetes, these modifications can require additional time compared to a traditional HPC environment. In reality, these participants experience a wider semantic distance. Furthermore, configuring YAML files is not a simple task, as discussed in Chapter 5.4.1.5.

Understanding the purpose of containerization itself is as important as understanding how it can benefit participants. This deeper understanding can help bridge the semantic distance that participants experience by providing more context behind the purpose of containerization.

Some participants believe that using a Kubernetes cluster works better than using a traditional HPC cluster for developing AI/ML models that are more sensitive to system performance variability, or noise, like weather simulations. In computing, noise refers to unpredictable variations in performance caused by factors like network latency, resource contention, hardware interruptions, or scheduling overhead [113]. These can lead to inconsistent job execution times or synchronization issues. Traditional HPC clusters are optimized for tightly coupled, resource-intensive workloads that require frequent synchronization among nodes, which can cause more noise. An infrastructure developer explains that OpenShift Kubernetes clusters are optimized for containerized, loosely coupled workloads, which are less dependent on frequent node synchronization:

(A13, Infrastructure Developer) “It’s pretty good for the AI model. For HPC, more sensitive to the noise, I still have doubts. For example, weather kind of simulation, you do synchronization a lot. So the network is definitely in OpenShift. It’s more noisy. But for AI models, this is not a problem.”

For data scientists using a Kubernetes cluster to develop AI/ML containerized apps, the underlying infrastructure is abstracted away to be hidden so that they don’t need to worry about those details. One of the goals of the infrastructure developers that work more closely with those details is to make things consistent across various clusters, such that users can easily migrate between clusters. Ideally, these abstractions help reduce the semantic and articulatory distance that users experience. This is in stark contrast to a user’s experience on a traditional HPC cluster, as an infrastructure explains that it requires all levels of users to have a deeper understanding of the underlying infrastructure:

(A13, Infrastructure Developer) *Of course, a lot of things are hidden, I mean, are done by the infrastructure. Once those are consistent across different clusters, then to the user, it's really much easier to migrate from one cluster, OpenShift, or Kubernetes cluster, to another. In HPC, it's definitely not the case. Different clusters will have different software packages. You may have to build your own."*

5.4.1.5 Configuring Jobs Using YAML

Unlike a traditional HPC cluster's simple SSH job submission, a Kubernetes cluster requires participants to execute more steps that involve building images and applying YAML files. Participants without prior Kubernetes experience, such as data scientists, struggle with YAML configurations, which widen the semantic distance that they experience. Infrastructure developer, cluster operator, and data scientist participants use Helm for job submission tasks, as shown in Appendix B in Tables B.8, B.7, and B.6. A data scientist explains that in order to abstract away more cumbersome tasks, such as applying YAML files, Helm templates are created by more experienced developers to be provided for them to use as a way to shorten the semantic distance they experience:

(A23, Data Scientist) *"That's great and why I'm able to use it, because if I had to do all the job by myself, that would be impossible. I would be using [organization A's traditional HPC cluster]. I'm using this image that someone created and somehow works. (...) I use Helm to convert this into, and again, the [redacted] people, they created this Helm template. I give this a smaller YAML and it will create the final one, which is huge."*

The OpenShift console provides participants with a GUI to ease job submission execution without the use of another tool, such as Helm. As an integrated dashboard, OpenShift offers participants more direct engagement. Participants also make use of scripts that automatically generate YAML files specific to the job they want to run from general templates in order to avoid

writing and updating them manually, further reducing the semantic distance they experience. A data scientist working on applications such as a PyTorch app-wrapper explain that YAML files can be large and complex as they must specify many parameters such as the kind of resources needed, the amount of CPUs and GPUs needed, and the amount of pods needed:

(A7, Data Scientist) “I had to do two things to build this template YAML with all the different parameters and then the Bash script that take the other configuration YAML to actually generate the specific YAML that I want to run. I did this one time (...) This is something which is on GitHub that I shared with a lot of folks and that seems to work pretty well with a lot of folks.”

Some issues persist with YAML that are related to an inability to fully integrate them into development, particularly due to versioning and configuration management issues. YAML files cannot be easily versioned in Git like source code because they contain sensitive data (e.g., secrets), which prevents their direct inclusion in Git repositories. As a result, manual versioning is required, which results in storing YAMLs in external systems like Box or sharing via Slack. This ad-hoc approach lacks the streamlined collaboration and history tracking provided by Git, making it harder to reuse or verify configurations as well as widening the semantic distance that participants experience. As a result, participants risk running jobs with incorrect settings due to untracked changes. YAMLs encapsulate critical parameters for AI/ML workloads such as training and inferencing and the absence of a standardized system for YAML versioning and sharing disrupts development. This is particularly painful in AI/ML containerized app development, where hyperparameter tuning is iterative and requires precise tracking to avoid errors. A library developer explains that the inability to fully integrate YAMLs into the development process due to their separation from source code, sensitivity to secrets, and lack of robust versioning creates inefficiencies, error-prone workflow:

(A2, Library Developer) *“YAMLs do not belong to the rest of the code. It’s sort of like, code and not code. (...) That has been the biggest challenge in some ways because people are like, ‘Look, I’m running the same thing.’ No, you are not. His argument to this function is different. You pass something else. And that portion of the code is not readily version controlled via Git. That we are doing manual version control. That’s the most painful part.”*

As explained by the library developer quoted above, the separate versioning processes for YAML and source code lead to challenges in collaboration, reproducibility, and configuration management for AI/ML containerized app development tasks on Kubernetes clusters.

5.4.2 Summary of Evaluation Gulfs

Overall, we found that participants experience distance when evaluating job submission tasks. The distance they experience is due to the lack of reliable experiment tracking and timely visibility into monitoring job progress, such as completion times, resource utilization, and outcomes. As well, participants experience tension between the necessary abstractions that Kubernetes introduces for usability. The abstractions they are faced with result in a loss of visibility into low-level issues.

Participants manually monitor and log their experiments, which is suboptimal for long-running AI/ML jobs as it is difficult to match configuration inputs with results. This difficulty widens the semantic distance that participants experience as they analyze their results across the many iterations of AI/ML experiments they run, which limits their ability to systematically compare them. Participants face reproducibility issues from forgetting input formats or hyperparameters, which leads to lost insights and repeated work. The semantic distance in the gulfs of evaluation that participants experience represents the difficulty of efficiently tracking and reproducing their AI/ML experiments using manual monitoring and logging methods.

Jobs frequently remain stuck in a pending state either because of the system’s computing

resource shortages, fragmentation, cluster instability, or because of the user’s misconfigurations. When these job failures or slowdowns go undetected, they result in delays in experiment tracking. These delays widen the semantic and articulatory distance that participants experience as they wait, unaware of, or unable to determine, their job progress. These feelings of semantic and articulatory distance are increased by the different levels of user privileges, that prevent participants from being able to independently determine whether their jobs are progressing efficiently, failing subtly, or wasting expensive resources. These feelings of semantic and articulatory distance are further increased by the limited resources available for education and support, which results in additional time spent stuck and reaching out for help from the few developers that either have more user privileges or a more integrated understanding of infrastructure-application knowledge. This is especially true for participants whose expertise lies outside of job submission tasks, which only exclude cluster operators, as shown in Figure 5.1.

These evaluation gulfs illuminate the gaps in default Kubernetes environments for AI/ML containerized app development-specific needs as well as drive the invention of more integrated automation tools to replace manual and fragmented development practices. The following provides a thematic analysis of the gulfs participants encounter in evaluating the outcomes of job submission tasks for AI/ML containerized app development, as listed the third row and second column of Table 5.1.

5.4.2.1 Managing Unpredictable Resource Availability

Unpredictable GPU availability and long scheduling delays (e.g., jobs stuck in pending) make it difficult to evaluate when or if jobs will complete. Scheduling unpredictability and resource contention obscure job progress, as participants cannot reliably predict completion times or detect failures promptly. A data scientist explains that jobs can remain pending for minutes, hours, or days due to GPU shortages, which hinders progress tracking and widens the semantic and articulatory distance that they experience:

(A24, Data Scientist) *“The biggest challenges, of course, the GPU availability. Many times we start the jobs, for some reason, things have not been very stable. I could do a helm install, I don’t see the PyTorch job scheduled at times. Even if it is stuck, it is always stuck in pending for minutes, or hours, or days together before it gets scheduled.”*

Evaluating whether jobs are optimally utilizing resources is difficult without clear metrics or guidelines, which impacts participants’ ability to optimize performance. Unclear resource utilization metrics make it hard to evaluate job efficiency and require manual monitoring and adjustments, widening the articulatory distance that participants experience. A library developer explains that inference optimization for foundation models requires them to have a deep knowledge of the underlying infrastructure that further widens the articulatory distance they experience:

(A2, Library Developer) *“I’m responsible for the inferencing, inference optimization, part of foundation models. The key thing to notice that a lot of these models, you can find a difference of 20 times, 10 times speed up with and without optimization. Which means you’re leaving a lot of money behind on the table if you are hosting and running unoptimized models and there isn’t one single bag, one single trick for optimizing a model. There is a bag of tricks. It is sort of art and science to how different tricks can be combined to help improve a model and it’s also a question of what’s the target hardware that you are looking at.”*

Participants emphasize the need for better documentation and development processes. They describe how existing documentation is insufficient and ranges from being nonexistent to overly comprehensive such that relevant information is difficult to find, widening the semantic distance that participants experience. As a result, participants, such as data scientists, reach out for help to further understand the documentation, widening the articulatory distance that they

experience. A data scientist explains that reaching out for help puts a pause on their progress and can lead to further confusion:

(A23, Data Scientist) *“Unfortunately, most of the time was trying to make sense of [organization A]’s documentation. (...) I have to, most of the time, talk to people and ask, get diverging information, and then go back and forth. I would say most of the time that I spent on learning how to use [organization A’s Kubernetes cluster] was to make sense of the documentation.”*

Participants with a limited knowledge of Kubernetes and related technologies experience a steep learning curve and struggle with troubleshooting. A lack of documentation for common issues leads to more experienced developers repeatedly helping those struggling. For example, a recurring problem is insufficient GPU availability which may be due to actual shortages of GPUs or incorrect configurations in YAML files that are preventing proper GPU allocation. Incorrect configurations could cause fragmentation, where GPUs are available but scattered across nodes, making them unusable for jobs requiring multiple GPUs. Diagnosing GPU issues requires iterative trial-and-error (e.g., running specific commands to check availability or configuration). Without documented such processes, this becomes a time-consuming and error-prone task that widens the semantic distance that participants experience. A project manager emphasizes that creating standardized documentation for troubleshooting and resource management would significantly alleviate issues of inefficiencies caused by limited team expertise and GPU-related challenges in the Kubernetes cluster:

(A21, Project Manager) *“That’s our biggest pain point of the present. Bringing in people who don’t know enough of these technologies, just having those blueprints that you can go after. ‘I’m running into this problem.’ Have you tried these five different things? Writing that blueprint down is just useful.”*

5.4.2.2 Ineffective Experiment Tracking

Tracking and comparing experiments is inefficient due to manual logging practices or non-integrated tools that make it hard to evaluate job outcomes. Participants struggle to systematically compare results or maintain accurate records and risk missing critical insights. These limitations widen the articulatory distance that participants experience. The Aim UI logs experiments, but with few experiments run due to long job durations, participants rely on manual graph comparisons rather than comprehensive logs. A data scientist explains that manual experiment logs on local computers are suboptimal, with challenges in cross-referencing notes to actual runs, highlighting the need for integrated tools such as MLflow to bridge the articulatory distance that they experience:

(A28, Data Scientist) *“I tried to keep an experiment log on my computer where I say, ‘Today I launched these experiments with these variables and these are the results.’ It’s definitely not optimal. It can be hard to match the actual experiment on my computer to what I’m referring to on my notes. I think this is where tools like MLflow can help because there’s no need for me to have my experiments on my computer and then my notes somewhere else. I have to be the one cross-referencing. This is definitely something that’s not optimal at the moment.”*

Infrastructure developers create more detailed dashboards (e.g., displaying resources consumed by specific app wrappers) for data scientists to use in order to evaluate past job utilization and help them independently make better informed, data driven decisions. An infrastructure developer explains that this helps data scientists to refine their allocations and shorten the articulatory distance that they experience:

(A25, Infrastructure Developer) *“This is my job and they can see how much utilization that needs and allocation they can go and adjust it. And the only way to do it before was, go to the cluster itself and they see the resources within the cluster, but the cluster shows you just by pods and namespaces. You can’t see specific app wrappers in there, but in this dashboard you can see the specific app wrappers and what resources the app wrappers are consuming.”*

Data scientists that run multiple AI/ML training experiments simultaneously lose track of configurations, such as input formats, hyperparameters, or specific settings, which causes reproducibility issues that widen the articulatory distance they experience. Without a standardized system for configuration management, they struggle to remember or reproduce exact experiment configurations (e.g., dataset formats, training parameters), which can lead to errors and inconsistencies. Managing experiments on a Kubernetes cluster requires handling complex configurations, which are not inherently versioned or easily reproducible in a default OpenShift Kubernetes platform. Data scientist teams need a reliable way to share and refer to experiment setups to ensure others could replicate results and avoid confusion such that their semantic and articulatory distance is shortened.

To address these challenges, data scientists invented a custom platform integrated with Git for a GitOps-like approach (i.e., MLOps) where experiment configurations are stored in a Git repository to ensure version control. They also make use of a customized Helm chart to automate job scheduling by defining parameters like the number of GPUs per pod, user, and experiment directory in a single YAML file that simplifies job submission. Other common tasks are abstracted into a Python library to reduce manual configuration and improve usability. Tools such as Aim provide experiment reports that provide more visibility for analysis. Custom MLOps platforms such as this ensure reproducibility, centralize configurations, and streamline experiment management. A data scientist explains how this allows them to track, share, and replicate AI/ML training jobs efficiently on the Kubernetes cluster:

(A25, Data Scientist) *“This is a platform that we have developed internally for managing our experiments. We run VS Code. You have the OpenShift console and stuff, you can use it for everything, but we want it in like a GitOps-like way. We call it [custom MLOps platform] within our team. We have multiples of trainings that we run, and when we started this project, we have multiple experiments to run and we were losing track of them. What we did was, we built this platform where we have all the experiments with all the configurations, everything they turn into Git. (...) For each of the models that we build or each of the trainings that we do, we have all the configuration required to reproduce it exactly as that, and this is the only way within the team we start the jobs or training. What this enables us is to make sure that anytime you develop a model, you might forget, ‘Did I give input in this particular format or that particular format. Did I give a space here?’ We might lose track of it, if we don’t have a prominence, so when we do everything when based on Git, we’ll go back to, you refer someone to this directory, you know exactly what is happening here. This is an application, the platform that we deliver, we run all the trainings on. (...) These are all driven by project that we have in internally where we have a Helm chart, which is customized for us, which allows us to say what are the exact configurations that are required for you to run. For example, for anybody to run a project to know about the cluster, all they need is only this [values.yaml]. They say how many pods needs to be run [numGPUsPerPod:] and who is running it [user:] and which folder we are running from [experimentsDir], and it’s all scheduled automatically. And we have things like Helm charts for Aim for looking at the reports and we have the, all the common things abstracted out as a Python library. This allows us to manage things, jobs running in [organization A’s Kubernetes cluster].”*

5.4.3 Job Submission Tools

Figure 5.10 provides a list of the 22 tools used for job submission tasks and illustrates that although one tool is used by most participants (i.e., oc, Helm, custom scripts), many other tools are also used. The “long tail” of tools indicates that participants require tools beyond oc, Helm, or custom scripts and as such, must spend additional time learning how to use them all. Similarly, figure 5.12 demonstrates that across all job roles, participants use many tools for job submission tasks. Figure 5.13 further demonstrates that across all job roles, participants use multiple tools for job submission tasks. Therefore, the “long tail” of tools appears in all categories of job roles and as such, everyone bears the burden of learning how to use and switch between multiple tools. More tables that break down the tools that participants use for job submission tasks by job role can be found in the Appendix B.

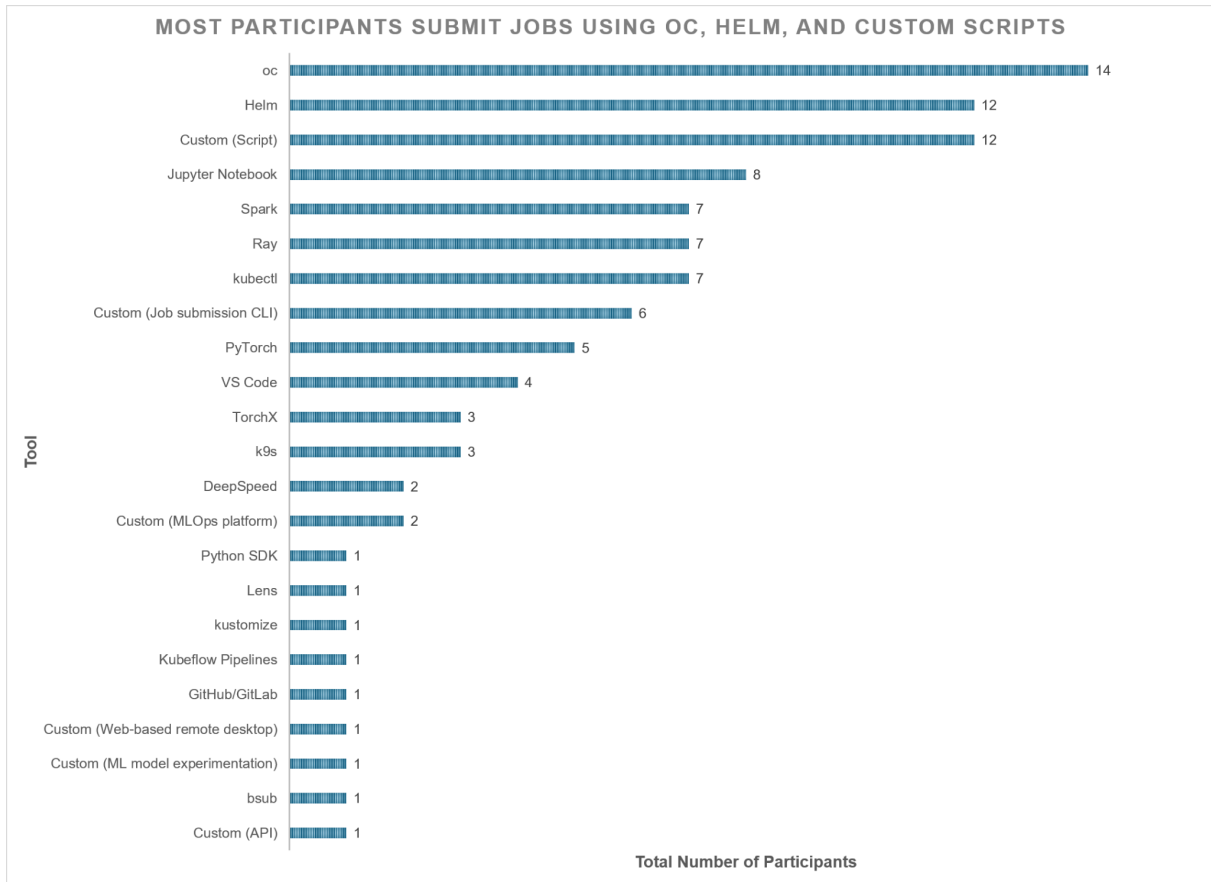


Figure 5.10. Overview of the tools used by participants for job submission tasks. Numbers represent the total amount of participants that use each tool.

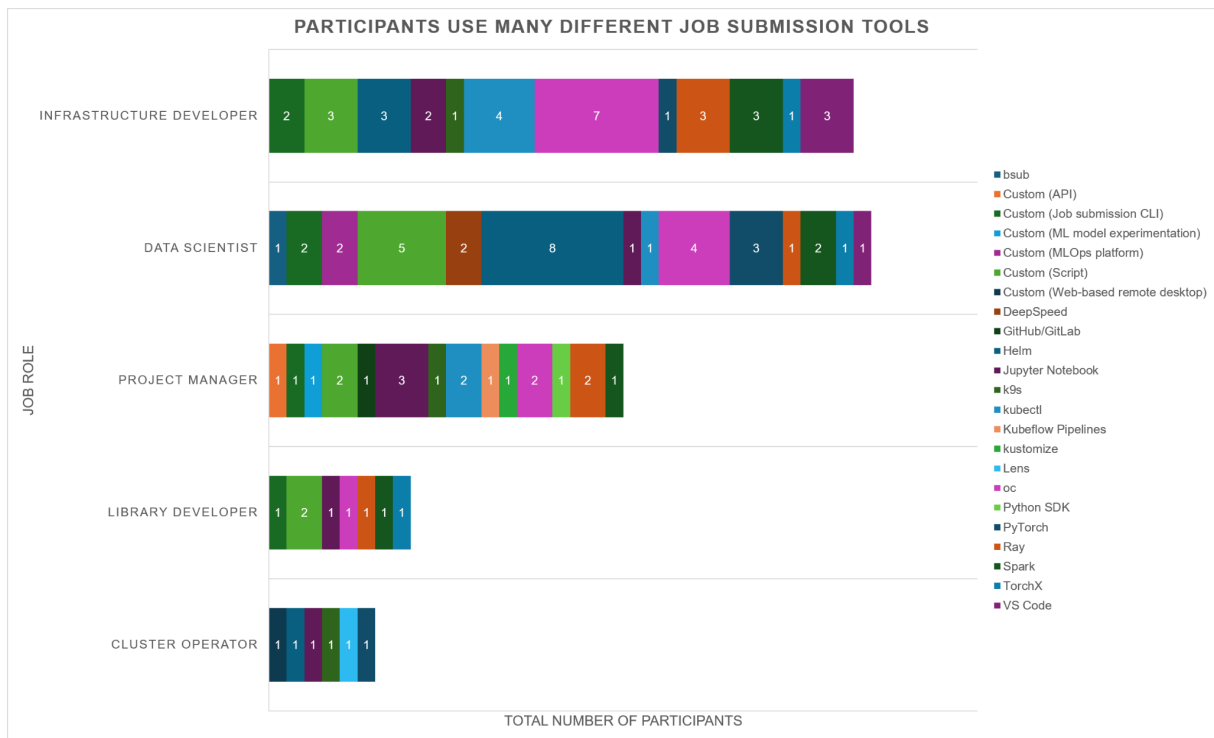


Figure 5.11. Breakdown of the tools used by participants with different job roles for job submission tasks. Colors represent unique tools and numbers represent the total amount of participants that use each tool.

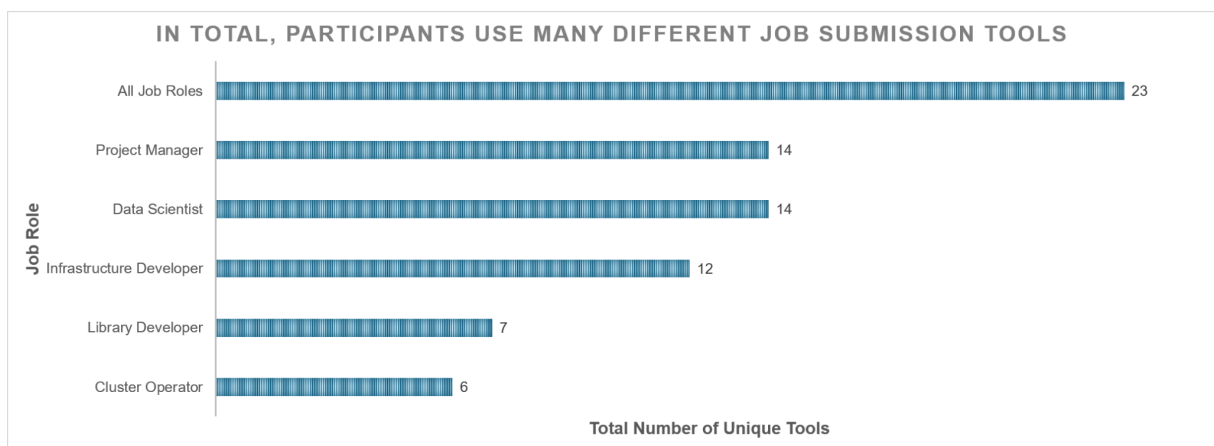


Figure 5.12. Overview of the total number of unique tools used by participants with different job roles for job submission tasks.

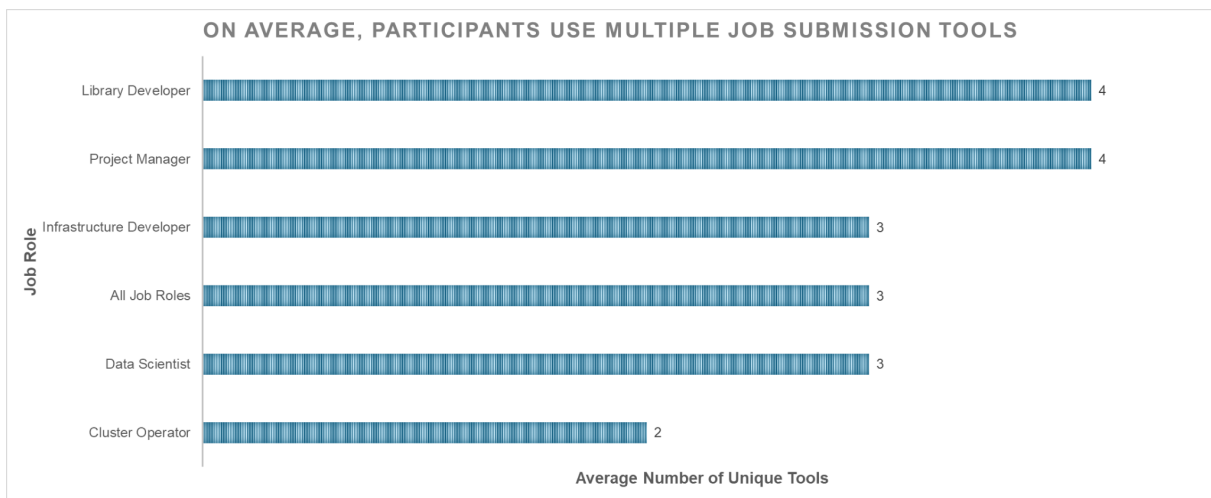


Figure 5.13. Overview of the average number of unique tools used by participants with different job roles for job submission tasks.

Chapter 6

Discussion

Our primary research question and hypothesis, stated in Chapter 4 are:

RQ. As people in larger teams often have distinct job roles with specialized task responsibilities that are intrinsic to those roles, to what extent does AI/ML containerized app development impact the separation of responsibilities among roles?

H. We hypothesize that AI/ML containerized app development, even as orchestrated through Kubernetes, undermines the separation of responsibilities among roles, leading to challenges due to a divergence from specialized tasks that requires developers to perform a multiplicity of new tasks and use tools outside of their specialization, resulting in increased communication with others for help.

In answering this research question, our results focus on the three tasks in which developers experience the most challenges: 1) dataset management, 2) debugging, and 3) job submission, as shown by the first, second, and sixth columns of Figure 5.1, respectively. Table 5.1 further breaks down Figure 5.1 by providing an overview of the specific challenges that contributed to the gulfs of execution and evaluation that developers reported experiencing when performing these three tasks.

Our results show how employing Kubernetes for AI/ML containerized app development can result in a developer performing tasks outside of their job role expertise, experiencing gulfs of

execution and evaluation as the distance from their expertise increases. Unfortunately, developers are frequently required to perform tasks outside of their expertise, revealing a lack of modularity in their job roles and a burden of additional responsibilities. By categorizing these challenges into gulfs, we are able to more clearly understand how a developer experiences points of friction in their workflow with regards to the specific development tasks they carry out via processes and tools.

A developer employing Kubernetes for AI/ML containerized apps must apply additional effort when managing their datasets, submitting a job run, and debugging their job run, as discussed in Chapters 5.2.1, 5.4.1, and 5.3.1, respectively. Likewise, that developer must apply additional effort when determining the state of their job run, including whether the cause of an error is due to either: 1) the way they've configured their job, 2) something outside of their control (e.g., user permissions), or 3) something outside of their awareness (e.g., system abstractions), as discussed in Chapters 5.2.2, 5.3.2, and 5.4.2, respectively.

Conflicts arise, such as an increased cognitive load when executing dataset management (see Chapters 5.2.1.1, 5.2.1.4, 5.2.2.1, and 5.2.2.2), debugging (see Chapters 5.3.2.1 and 5.3.2.4), and job submission (see Chapter 5.4.1.3) tasks. An increase in cognitive load acts as a burden, resulting in the need to spend additional time performing these tasks and eventually, the need to invest effort into creating custom tools to help abstract away or automate the responsibilities that come along with these tasks, as discussed in Chapters 5.2.2.2, 5.3.2.1, 5.3.2.3, 5.4.1.1, 5.4.1.3, 5.4.1.5, and 5.4.2.2. Imbalances also arise, such as an inability to continue making progress, which results in more time spent reaching out for help from a more experienced developer, whose progress is also interrupted by the need to help a novice developer resolve their issue, as discussed in Chapters 5.2.1.2, 5.2.2.3, 5.3.2.1, 5.3.2.3, 5.3.3, 5.4.1.1, 5.4.1.2, and 5.4.2.1.

We now discuss the implications of our results, reflecting on three specific questions in relation to our primary research question:

1. What are the most frequently reported challenges that AI/ML containerized app developers

experience?

2. Which AI/ML containerized app developers report experiencing the most challenges?
3. To what extent are these challenges unique to AI/ML containerized app development?

6.1 What are the Most Prevalent Challenges?

Quantitatively, Figure 5.1 reveals the extent to which participants reported to experience the qualitative challenges shown in Table 5.1. In Table 6.1, we further break down these qualitative challenges to understand their prevalence in employing Kubernetes for AI/ML containerized app development. Although Figure 5.1 captures the percentage of participants that reported in at least one response that they experience gulfs of execution or evaluation when performing a development task, Table 6.1 captures the percentage of participant responses that indicate the specific challenges within these gulfs. As participant responses could reference multiple challenges within them, we attempted to group them based on the dominant challenge discussed, with consideration for the context of the conversation (i.e., the question asked to prompt their response and general flow of the conversation surrounding their response).

Table 6.1. The percentage of participant responses that indicate they experienced a specific challenge, identified within the gulfs of execution and evaluation for a given task. “Other” represents the percentage participant reports that did not fit into any of these specific challenges.

Task	Execution Challenges (Percent of Participant Responses)	Evaluation Challenges (Percent of Participant Responses)
Dataset Management	• Mounting storage and managing credentials using COS (37%) • Dealing with large datasets (31%) • Continuous optimization of infrastructure and storage (24%) • Ensuring data quality through complex filtering processes (8%)	• Overcoming the learning curve for COS (38%) • Infrastructure-application knowledge (37%) • Adapting storage from HPC to Kubernetes (25%)
Debugging	• Adapting debugging from interactive to batch sessions (52%) • Overcoming cluster unreliability with checkpointing and iterative debugging (26%) • Manual process of logging (12%) • Other (10%)	• Limited visibility and uncertainty of an error's root cause (43%) • Inadequate tooling for log analysis (19%) • Balancing observability and alert fatigue (14%) • Multi-expert, real-time coordination (14%) • Other (10%)
Job Submission	• Computing resource contention and manual coordination (48%) • Infrastructure variability and hybrid cloud abstractions (30%) • Users with varying experience and access (11%) • Switching among multiple tools due to a lack of integration (9%) • Configuring jobs using YAML (2%)	• Managing unpredictable resource availability (70%) • Ineffective experiment tracking (26%) • Other (4%)

Dataset management, debugging, and job submission tasks are defined in Table 4.9.

Dataset management and debugging tasks are attributed to the expertise of infrastructure and

library developers. Job submission tasks are attributed to the expertise of cluster operators. Participant job roles are defined in Table 4.8.

Regarding executing dataset management tasks, the most prevalent challenges include: 1) mounting storage and managing credentials using COS (37%), 2) dealing with large datasets (31%), and 3) the continuous optimization of infrastructure and storage (24%). Our findings regarding the nature of these challenges are discussed in Chapters 5.2.1.4, 5.2.1.1, and 5.2.1.3, respectively. Regarding evaluating dataset management tasks, the challenges are more evenly distributed between the three themes we identified, as shown in the first row and second column of Table 6.1. Our findings regarding the nature of these challenges are discussed in Chapter 5.2.2.

Regarding executing debugging tasks, the most prevalent challenge is adapting debugging from interactive to batch sessions (52%). Our findings regarding the nature of this challenge are discussed in Chapter 5.3.1.1. Regarding evaluating debugging tasks, the most prevalent challenge is the limited visibility and uncertainty of an error's root cause (43%). Our findings regarding the nature of this challenge are discussed in Chapter 5.3.2.3.

Regarding executing job submission tasks, the most prevalent challenges include: 1) computing resource contention and manual coordination (48%) as well as 2) infrastructure variability and hybrid cloud abstractions (30%). Our findings regarding the nature of these challenges are discussed in Chapters 5.4.1.1 and 5.4.1.4, respectively. Regarding evaluating job submission tasks, the most prevalent challenge is managing unpredictable resource availability (70%). Our findings regarding the nature of this challenge are discussed in Chapter 5.4.2.1.

Together, the gulfs in Figure 5.1 and the extent of the challenges presented by these gulfs in Table 6.1, reveal three repeated patterns across dataset management, debugging, and job submission tasks: 1) a divergence from specialized tasks, 2) switching between a multiplicity of tools, and 3) increased communication with others for help, increasing the distance that developers experience. Below, we further discuss the implications of these patterns.

6.1.1 A Divergence From Specialized Tasks

Regarding the first pattern, participants reported that they frequently perform tasks that are beyond their expertise, as discussed in Chapter 5, with particular detail in Chapters 5.2.2.3, 5.3.2.1, 5.3.2.3, 5.4.1.2, 5.4.1.4, and 5.4.2.1. Figure 5.1 supports this, showing that participants experience a high level of non-modular work, such that, with few exceptions, everyone is doing a bit of everything. Notably, dataset management, debugging, and job submission tasks are not only commonly performed across most job roles as well as outside of participants' defined expertise, but they also present significant challenges in terms of the gulfs of execution and evaluation.

Regarding evaluating dataset management tasks, as shown in the first row and second column of Table 6.1 and discussed in Chapter 5.2.2.3, infrastructure-application knowledge (37%) entails a full-stack understanding, which in the context of employing Kubernetes for AI/ML containerized app development, requires knowledge in multiple domains (i.e., AI/ML model applications and the infrastructure they are deployed on). It may be uncommon for people to have all of these skills and be experts in multiple domains. Indeed, this is true for many organizations that have defined job role specializations and functional teams, which serve to increase the modularity and productivity of the work they perform, as explained in Chapter 4. Although such organizations hope to not require people to have all of these skills, in practice, we found that developers experience this challenge when employing Kubernetes for AI/ML containerized app development. Furthermore, our interviews revealed how there is a limited number of experts with sufficient infrastructure-application knowledge and that this limited number is shrinking, not growing, as discussed in Chapter 5.4.1.2.

Regarding evaluating debugging tasks, as shown in the second row and second column of Table 6.1 and discussed in Chapter 5.3.2.1, multi-expert, real-time coordination (14%) results, in part, from a lack of full-stack expertise. Similarly, Kubernetes and OpenShift abstractions make it difficult for participants to understand job statuses, performance impacts, or failure causes

without full-stack expertise. Furthermore, as shown in the second row and second column of Table 6.1 and discussed in Chapter 5.3.2.3, limited visibility and uncertainty of an error's root cause (43%) due to these abstractions also contribute to the complexity faced by participants when debugging a failure with their job run or the cluster itself.

Regarding executing job submission tasks, as shown in the third row and first column of Table 6.1 and discussed in Chapter 5.4.1.4, infrastructure variability and hybrid cloud abstractions (30%) require all levels of users to have a deeper understanding of the underlying infrastructure and hence, full-stack expertise. Furthermore, as shown in the third row and first column of Table 6.1 and discussed in Chapter 5.4.1.2, users with varying experience and access (11%) entails limited user privileges that create a hierarchy where users with less privileges must depend on those with more. This worsens the challenges described above regarding dataset management, debugging, and job submission tasks by severely limiting independent work, slowing high-stakes AI/ML containerized app development under tight deadlines.

Regarding evaluating job submission tasks, as shown in the third row and second column of Table 6.1 and discussed in Chapter 5.4.2.1, managing unpredictable resource availability (70%) is a result of the high computing resource demands from competing AI/ML jobs. This creates computing resource contention, further worsening issues of delayed feedback and interruptions from other tasks, such as dataset management and debugging. These issues are compounded by the learning curve that novices face regarding making the decisions that come with the additional steps of employing Kubernetes for job submission tasks, such as computing resource estimation, configuration, and provisioning, as such decisions are outside of their expertise.

6.1.2 Switching Between a Multiplicity of Tools

Regarding the second pattern, participants reported that they make use of a multiplicity of tools to help bridge the gulfs they experience, as discussed in Chapter 5, with particular detail in Chapters 5.2.1.1, 5.2.2.1, 5.3.1.2, 5.3.1.3, 5.3.2.1, 5.3.2.3, 5.4.1.1, 5.4.1.3, 5.4.1.4, and 5.4.2.2. The tools that participants reported using include both off-the-shelf and custom tools,

invented by the participants themselves. Furthermore, Chapters 5.2.3, 5.3.3, and 5.4.3, reveal that, across all job roles, participants on average use multiple tools for dataset management, debugging, and job submission tasks, respectively. Therefore, everyone bears the burden of learning how to use and switch between multiple tools to perform these three tasks.

Regarding dataset management tasks, Chapter 5.2.3 discusses Figures 5.2, 5.4, and 5.5, which provide an overview of the different tools that participants use. Regarding the gulfs of evaluating dataset management tasks, as shown in the first row and second column of Table 6.1 and discussed in Chapter 5.2.1.1, dealing with large datasets (31%) is a challenge that may involve starting small to experiment before scaling up, for which participants may address using various tools and libraries.

Regarding debugging tasks, Chapter 5.3.3 discusses Figures 5.6, 5.8, and 5.9, which provide an overview of the different tools that participants use. Regarding the gulfs of executing debugging tasks, as shown in the second row and first column of Table 6.1 and discussed in Chapter 5.3.1.2, overcoming cluster unreliability with checkpointing and iterative debugging (26%) is a challenge that has been addressed, in part, by using tools such as checkpoints. Participants reported that using checkpoints is a necessary measure to resume progress on large-scale, long-running AI/ML job runs that are interrupted due to cluster unreliability, saving them time from not having to restart their job run from the beginning. However, using checkpoints is an additional step that participants who are unfamiliar with Kubernetes must learn how to effectively implement.

Regarding the gulfs of evaluating debugging tasks, as shown in the second row and second column of Table 6.1 and discussed in Chapter 5.3.2.1, multi-expert, real-time coordination (14%) is a challenge that has been addressed, in part, by the custom tools that participants created in order to minimize the additional communication they face in having to coordinate with multiple experts when debugging issues with their job run or the cluster itself. Furthermore, as shown in the second row and second column of Table 6.1 and discussed in Chapter 5.3.2.3, limited visibility and uncertainty of an error's root cause (43%) is a prevailing challenge despite all of

the well-designed tools available to use off-the-shelf and the custom tools invented by developers themselves. Participants reported that there is no simple solution when it comes to evaluating the issues of a Kubernetes cluster and that they and their team are constantly dealing with new issues.

Regarding job submission tasks, Chapter 5.4.3 discusses Figures 5.10, 5.12, and 5.13, which provide an overview of the different tools that participants use. Regarding the gulfs of executing job submission tasks, as shown in the third row and first column of Table 6.1 and discussed in Chapter 5.4.1.1, computing resource contention and manual coordination (48%) is a challenge that could be better handled. Participants reported that in order to resume progress when their job is terminated due to computing resource contention, they must seek out other environments to run their jobs and apply additional effort to adapt their jobs to run in that different environment. Similarly, as shown in the third row and first column of Table 6.1 and discussed in Chapter 5.4.1.3, switching among multiple tools due to a lack of integration (9%) is, in part, due to such job terminations. To address these challenges, participants reported using custom scripts to automatically adapt configurations when switching between deploying their job in a Kubernetes and traditional HPC environment.

Regarding the gulfs of evaluating job submission tasks, as shown in the third row and second column of Table 6.1 and discussed in Chapter 5.4.2.2, ineffective experiment tracking (26%) is a challenge that has been addressed, in part, by the custom platforms that participants invented to integrate with Git for a GitOps-like approach, ensuring reproducibility, centralizing configurations, and streamlining experiment management. This challenge is also related to one of the gulfs of executing debugging tasks, as shown in the second row and first column of Table 6.1 and discussed in Chapter 5.3.1.3, manual process of logging (12%). Manual logging practices and non-integrated tools make it hard to execute debugging and evaluate job outcomes as participants reported that they struggle to systematically compare results or maintain accurate records, highlighting the need for integrated tools.

6.1.3 Increased Communication With Others for Help

Regarding the third pattern, participants frequently reported that they reach out for help to resolve issues, as discussed in Chapter 5, with particular detail in Chapters 5.2.2.3, 6.1.1, 5.3.2.1, 5.3.2.3, 5.4.1.2, and 5.4.1.1. More specifically, Chapter 5.3.3 reveals that the most used tool for debugging is a communication tool that participants use to message others for help. The third and fourth most used tools also regard means for communicating with others for help. The extent to which participants reach out for help further reveals a lack of modularity in their development process, as independent work becomes impossible and additional communication becomes necessary.

Regarding evaluating dataset management tasks, as shown in the first row and second column of Table 6.1 and discussed in Chapters 5.2.2.3 and 6.1.1, one of the most prevalent challenges we identified is regarding infrastructure-application knowledge (37%). However, this challenge is also relevant to debugging and job submission tasks. Without a full-stack understanding, all three of these tasks are more difficult to independently accomplish.

Regarding evaluating debugging tasks, as shown in the second row and second column of Table 6.1 and discussed in Chapter 5.3.2.1, multi-expert, real-time coordination (14%) reinforces that many of the developers we interviewed reported that they lack a full-stack understanding. As a result, they become stuck and must reach out for help from the few who have such knowledge, waiting for help from others in order to resume progress. Furthermore, as shown in the second row and second column of Table 6.1 and discussed in Chapter 5.3.2.3, limited visibility and uncertainty of an error's root cause (43%) reveals how debugging jobs on Kubernetes clusters is inherently complex due to the involvement of multiple components that require varying job role expertise. Due to this, participants with different job roles must depend on each other's expertise to perform debugging in a combined effort, requiring frequent coordination that is time-consuming and may be disruptive.

Regarding executing job submission tasks, as shown in the third row and first column

of Table 6.1 and discussed in Chapter 5.4.1.2, users with varying experience and access (11%) further requires participants with different job role expertise and cluster privileges to depend on one another, limiting their independent work. This further depends on the availability of computing resources and the person who possesses the cluster privileges necessary to resolve the issue. Therefore, additional time and delays may result from the required back-and-forth communication throughout the process of submitting a job. Furthermore, as shown in the third row and first column of Table 6.1 and discussed in Chapter 5.4.1.1, computing resource contention and manual coordination (48%) further worsens issues of delayed feedback and interruptions from other tasks, such as dataset management and debugging. Manual coordination also results from the additional negotiations with cluster admins that is required to secure large, high-priority computing resource allocations, which is time-consuming.

6.2 Who Experiences the Most Challenges?

Interestingly, Figure 5.1 shows that performing AI/ML containerized app development tasks generates non-modular work for everyone, regardless of their job role. This finding directly builds on an initial survey we conducted and presented in prior work, discussed in Chapter 3.3.2.1, where we found that the challenges of employing Kubernetes in this context largely stem from the need for everyone to know everything, requiring a full-stack understanding [80]. Despite interviewing the most participants from data scientist job roles (13), Figure 5.1 shows that it is not only the data scientists that experience challenges. All participants, with the exception of library developers, reported performing all tasks.

Library developers did not report performing monitoring and cluster observation tasks, as shown in the fourth and seventh columns of the fourth row in Figure 5.1. In part, this may be due to the fact that developing libraries for AI/ML doesn't require as much active monitoring of a job or observation of the cluster, compared to the tasks performed by other job roles. As well, this may be due to a difference in the total number of participants we interviewed from each of

these job roles, as we interviewed only two library developers. However, we only conducted interviews with 35 participants and therefore, did not interview many people overall. Therefore, our findings regarding who experiences the most challenges are a starting point for future work to consider.

Although the gulfs experienced by our participants may be different depending on their job role, they are entangled in a manner that restricts the modularity of their work. For example, a data scientist that reaches out for help from an infrastructure developer to debug their job performance because they do not have enough independent understanding. As a result, that infrastructure developer may be interrupted to help the data scientist.

6.3 To What Extent are These Challenges Unique to Containerized Development?

The extent to which the challenges we identified are unique to containerized development or AI/ML is especially relevant as we found that many of our participants conducted AI/ML containerized app development in a traditional HPC environment prior to employing Kubernetes. Therefore, understanding the specific challenges AI/ML containerized app developers newly face in employing Kubernetes is important. To do so, we discuss our results in view of the large-scale analyses, surveys, and user studies discussed in Chapters 3.3.1.1, 3.3.1.2, 3.3.1.3, 3.3.2.1, and 3.3.2.2, which provide a mix of quantitative and qualitative results regarding general and AI/ML containerized app development. As our results have implications for future tool development, we also discuss their implications in view of the tool interventions for AI/ML containerized app development discussed in Chapter 3.3.2.2, which provides more quantitative results.

Although these studies focus on containerized developers with job roles similar to those found in our study, there are some differences in title. This is partly due to how we identify the tasks that correspond to developers' job roles, as defined in our initial survey, presented in prior

work by Arroyo et al. [80]. This could also be because our participants, despite being within the same community of containerized developers, likely come from different organizations that have their own defined job roles.

In order to better understand the results of our study in the context of these prior works, we mapped their job roles to those in our own study, as shown in Table 6.2. We mapped up their job roles to our own based on how we define them as well as their specialized task responsibilities, as shown in Tables 4.8 and 4.9. We do so to the best of our understanding, as these prior works do not always clearly define the job roles or explain the development tasks performed within them. We omitted some of the job roles from the table as we did not interview developers with those job roles. Omitted job roles from the 2025 CNCF survey include professional services/consultant, executive, network admin, and teacher [33]. Omitted job roles from the Zhou et al. interviews include CTO, product owner, and software architect [211].

Table 6.2. Proposed job role correspondences from prior works that focus on similar job roles. “-” refers to the job roles in prior work that do not directly correspond to our own.

Prior Work	Cluster Operator	Data Scientist	Infrastructure Developer	Library Developer	Project Manager
2025 CNCF survey [33]	Systems Admin	Developer/Engineer Scientist/Researcher/Specialist	Developer/Engineer	Developer/Engineer	Manager/Director/VP
SnapperML framework [164] Kubeflow service [129] DEEP Framework [125] Lee et al. platform [149]	-	Data Scientist [164, 129, 125] ML Developer [149]	-	-	-
Scanflow-K8s framework [156] Katib framework [127]	Data Engineer [156] Operations Engineer [127]	Data Scientist	-	-	-
Zhou et al. interviews [211]	-	Software Developer	Software Developer	Software Developer	Manager
OCDL platform [157]	-	App Developer Data Scientist	App Developer	App Developer	Project Manager
2018 SIG Apps survey [120]	App Operator Cluster Operator	App Developer Tool Developer	App Developer Tool Developer	App Developer Tool Developer	-

6.3.1 Challenges in General Containerized Development

Similar to the first two patterns identified in our findings discussed in Chapters 6.1.1 and 6.1.2, a large-scale analysis and survey found that general containerized development involves a full-stack understanding such that developers must possess knowledge in multiple domains [135], effectively performing several job roles using a multiplicity of tools [120, 119]. Similar to the third pattern discussed in Chapter 6.1.3, a user study found that developers employing DevOps and microservices experience increased communication with others for help, as they must frequently collaborate across multiple teams with different expertise in order to resolve issues [211]. Similar to the most prevalent challenges regarding dataset management tasks shown in Table 6.1 and discussed in Chapter 6.1, tool interventions found that developers struggle to adapt their work from traditional HPC to Kubernetes environments, as it requires them to perform tasks beyond their training [155, 111]. These prior works did not go into detail regarding the types of projects that the developers they studied were working on. Therefore, it may be the case that some were working on projects in the AI/ML domain. However, we regard them as general containerized development studies as, to our knowledge, they do not focus on the context of AI/ML.

6.3.1.1 Large-Scale Analyses

As discussed in Chapter 3.3.1.1, Haque et al. focus on the challenges faced by developers employing container technologies, such as Docker [135]. Our interviews with AI/ML containerized app developers, who use Docker along with Kubernetes, revealed how using Docker is part of the learning curve that they experience. An infrastructure developer with more than 20 years of experience using Linux explains that the learning curve for Kubernetes takes time to overcome and involves learning Docker:

(A3, Infrastructure Developer) “[Learning Kubernetes] takes some time. Imagine for myself, for example, I started [using Linux] at the age of 17-ish. It’s like 20 years or even more [since I started using Linux]. Don’t expect to learn Kubernetes in two weeks, no. Take your time. Learn Docker before you learn this. Nearly each day, I’m learning something new.”

Haque et al. found that that Docker experts are scarce among the respondents to Stack Overflow (SoF) posts on Docker development, attributing this to how being a Docker expert requires knowledge in multiple domains (i.e., networking, operating systems, cloud computing and software engineering) [135]. In the context of employing Kubernetes for AI/ML containerized app development, not only did our participants report that they must have such Docker expertise, but that they also require further knowledge in AI/ML model applications and the infrastructure they are deployed on.

6.3.1.2 Surveys

As introduced in Chapter 3.3.1.2, our study builds on the SIG Apps surveys, which focus on Kubernetes tasks and tools outside the context of AI/ML containerized app development [120, 119]. Similar to our study, in the 2018 SIG Apps survey, a majority of respondents reported performing multiple job roles, identifying themselves as cluster operators along with other roles [120]. As a reminder, although the 2018 SIG Apps survey distinguishes between application operator and cluster operator job roles in a more general context, we do not report on application operators as a separate job role in our study. Unfortunately, they do not further define these job roles or explain the development tasks they perform. Therefore, to the best of our understanding, we correspond them to our own, as shown in the last row of Table 6.2.

In our interviews, all developers, despite their job role, reported performing “Operator” tasks, such as job submission, as shown by the orange bars that reach 100% for each row in the “Job Submission” task column in the middle of Figure 5.1. Given that the 2018 SIG Apps

survey found that respondents most frequently perform “Operator” tasks (i.e., which include job submission) along with other tasks, this mirrors what we found to be one of the most challenging tasks, and emphasizes its importance. Furthermore, our interviews revealed that most participants experience gulfs of execution and evaluation when performing job submission tasks, as shown by the green and blue bars in the “Job Submission” task column in the middle of Figure 5.1. In fact, only cluster operators did not report experiencing gulfs of evaluation for job submission tasks, which could be due to the fact that they are attributed as the only job role with expertise in this task.

Similar to our study, the 2018 SIG Apps survey also found that respondents use a multiplicity of tools, including their own custom tools [120]. In further similarity to our study, they found little overlap among the tools that respondents reported using, which suggests the existence of many alternatives and a lack of standard practices [120]. Regarding these tools, they found that there is a need for custom tooling, data visualizations, and practical documentation for containerized developers [120]. Our interviews also revealed the need for more integrated data visualizations and how using a multiplicity of tools reflects the semantic and articulatory distance in the gulfs of evaluation that containerized developers experience.

Our study builds on the findings presented by the SIG Apps surveys and show how tools (e.g., custom tools, data visualization integrations) relate to job role expertise, which these survey did not. Earlier, in Chapter 6.1.1, we discussed how a full-stack understanding is detrimental to the modularity that job roles provide for development teams. Our interviews revealed the challenges that prompt containerized developers to build or use custom tools as well as create educational materials. For example, to help bridge gulfs of evaluation that developers experience when they lack a full-stack understanding.

6.3.1.3 User Studies

As introduced in Chapter 3.3.1.3, Zhou et al. conducted interviews with developers to understand the challenges they face employing DevOps and microservices [211]. Our interviews

with AI/ML containerized app developers, who perform DevOps tasks, revealed how it's part of the learning curve that they experience. A library developer shares an example regarding someone they know on a different team who does not experience as steep of a learning curve because they do not have to "worry" about becoming proficient in Cloud Object Storage (COS) and other DevOps tasks, as it is already set up for them:

(A6, Library Developer) "I think he was saying something along the lines of (...) when a new researcher was onboarded, they got their own COS with some minimum number of storage, one terabyte or something. Basically, the researcher didn't have to worry about any of the COS stuff or any of the DevOps-related things."

However, the situation explained in the above quote is not a typical one. In our interviews, most developers reported that learning how to use COS is a struggle and this is especially true for data scientists. An infrastructure developer explains how DevOps is a new mindset that data scientists are unfamiliar with and must learn along with using an OpenShift Kubernetes platform, as it is outside of their expertise:

(A13, Infrastructure Developer) "I think for [data scientists that are struggling with learning Kubernetes it's] because they don't have so much software experience. I'm coming from a software background, so I know [the DevOps mindset] inside and out. But as I work with [data scientists], I can see definitely it's because it's a totally different area. Their domain is in AI/ML model training. Of course they don't know all those details. When you move to OpenShift, [that] is something you do have to learn."

Zhou et al. found that in practice, the DevOps development cycle fragments operations and deployment tasks such that infrastructure operations are independent from their development, creating an organizational structure with frequent meetings that require many experts from different teams to join in order to solve the problems encountered [211]. This contrasted with the

promised benefits of employing DevOps and microservices, such as enhanced developer abilities and skills that result from the act of switching between job roles and learning different expertise, as teams must be responsible for tasks across the entire development cycle of DevOps.

Our interviews revealed that in the context of AI/ML containerized app development, developers across all job roles must perform tasks that span the entire development cycle. Similar to Zhou et al.'s findings, performing tasks outside of one's expertise results in developers reaching out to others with more experience or different expertise, as well as frequent meetings between teams with different expertise, to solve the problems encountered. Ideally, the separation of infrastructure operations from application development can help AI/ML containerized app developers focus on their specialization. However, in practice, they still must perform and evaluate tasks beyond their expertise. Furthermore, for the most part, our participants experience gulfs of execution and evaluation when performing tasks outside of their expertise.

6.3.1.4 Takeaways on General Containerized Development

As seen in these prior works, the wider containerized development community is reporting that requiring such a full-stack understanding is wishful thinking. Although most containerized developers perform multiple job roles [120, 119], in practice, a lack of Docker experts signals that there are few who possess a full-stack expertise [135]. As well, DevOps and microservices developers frequently collaborate across multiple teams with different expertise in order to resolve issues [211]. Our interviews also revealed that such a full-stack understanding is uncommon, perhaps due to organizations having specialized job roles and functional teams, as explained in Chapter 4. Therefore, as specializations are more typical, this implies that developers frequently perform tasks outside of their expertise. Furthermore, the wider containerized development community is reporting that developers adapting their workloads from traditional HPC to Kubernetes environments struggle with the additional, time-consuming tasks they must learn that are outside of their expertise [155, 111].

Although we conducted our study in a more specific context than these prior works,

AI/ML containerized app development is still a relatively new use for Kubernetes, as discussed in Chapters 3.3.1.2 and 3.3.2.1. Therefore, these prior works tell us that the gulfs we found may be widely experienced in the early stages of employing Kubernetes for AI/ML containerized app development. Furthermore, the quantitative results contributed by the large-scale analysis and survey provide a gauge for the significance of the qualitative challenges faced by the developers we interviewed.

6.3.2 Challenges in AI/ML Containerized App Development

Similar to the findings of the prior works on general containerized development discussed above, other surveys found that AI/ML containerized app development involves a full-stack understanding such that developers perform multiple job roles using a multiplicity of tools [80] and face challenging organizational and cultural changes [33]. In further similarity, tool interventions present solutions that address, in part, issues that both general and AI/ML containerized app developers face in performing tasks that diverge from their expertise [149], including increased communication with others for help [127, 156, 157, 129]. Other tool interventions address challenges that are unique to AI/ML containerized app development, such as experiment tracking [164] and dealing with large-scale datasets [125].

6.3.2.1 Surveys

As introduced in Chapter 3.3.2.1, our study builds on the CNCF annual surveys [13, 8, 4], which more broadly focus on Kubernetes tasks and tools. Similar to our study, the 2025 and 2024 CNCF surveys distinguish among the types of AI/ML containerized app development tasks that respondents report using Kubernetes for [33, 13]. Our interviews revealed that overall, most AI/ML containerized app development effort is spent on the dataset preprocessing stages, as ensuring the quality of a continuous collection of new, larger datasets adds repeated effort. Dataset preprocessing is mentioned as one of the AI/ML development tasks reported on in the 2025 and 2024 CNCF surveys, which further emphasizes the importance of this task [33, 13].

Although the 2025 and 2024 CNCF surveys report more broadly on the most prevalent challenges faced in AI/ML containerized app development (i.e., organizational and cultural changes), they do not report on the reasons behind or explain the challenges of employing Kubernetes for AI/ML containerized app development. This qualitative data was revealed in our interviews, further identified as gulfs of execution and evaluation, and is supported by the quantitative data provided by these CNCF surveys.

Similar to the SIG Apps surveys which focus on general containerized development, as discussed in Chapter 3.3.1.2, the CNCF surveys do not compare how tools relate to different job roles and development tasks. Although this makes it difficult for us to compare our results, in Chapter 4.3, we define the job roles and tasks that AI/ML containerized app developers perform within their expertise, as well as identify the tools they use to perform these tasks. Therefore, we provide some of the missing context regarding the job roles and tools that these surveys report on, which contributes to a clearer understanding of their results. Furthermore, by defining the tasks that fall within a job roles, we can clearly identify the tasks developers perform outside of their job roles, revealing added friction in their workflows.

As introduced in Chapter 3.3.2.1, our study directly builds on an initial survey we conducted and presented in prior work by Arroyo et al. [80]. Similar to our study, this survey found that the challenges of employing Kubernetes in this context largely stem from the need for everyone to know everything, requiring a full-stack understanding. This can be an overwhelming experience and reflects similar findings in the user study on DevOps and microservices development by Zhou et al. [211]. Similar to our study and the 2018 SIG Apps survey [120], discussed in Chapter 3.3.1.2, this survey found that all respondents assume cluster operator tasks and most frequently perform multiple job roles [80]. Again, the quantitative results contributed by these surveys provide a gauge for the significance of the qualitative challenges faced by the developers we interviewed.

This initial survey also found that the respondents performing multiple job roles must frequently switch between different tasks related to deployment, dependency management,

tracking cluster health, and coordinating job submissions [80]. However, unlike our study, this survey does not define and correlate the job roles and tasks that AI/ML containerized app developers perform within their expertise. We are able to provide some of the missing context regarding the job roles and tasks that this survey reports on by defining them in our study, which contributes to a clearer understanding of their results.

6.3.2.2 Tool Interventions

As introduced in Chapter 3.3.2.2, prior work on tool interventions present solutions that represent their hypotheses regarding what challenges people experience in the context of AI/ML containerized app development [149, 157, 125, 127, 129, 156, 164]. However, as the focus of these studies was to build a tool, only four of them included case studies to support their tool intervention [149, 157, 156, 164]. None of them conducted a user study to support their tool intervention as a proof of concept.

Our study adds context and further evidence for the efficacy of these prior tool intervention studies by conducting interviews to understand the perspectives of AI/ML containerized app developers. These tools have been built to solve problems and attempt to present better paths to the future of employing Kubernetes for general and AI/ML containerized app development. Our interviews contribute further insight on how such tools can be prioritized and improved for AI/ML containerized app development. By understanding the challenges addressed by tools, we also contribute further insight on what challenges are more generally experienced by developers employing Kubernetes, as opposed to what challenges are more specifically experienced by developers employing Kubernetes in the domain of AI/ML.

Dealing With Large Datasets.

Garcia et al. built DEEP, a Kubernetes framework to address the challenges data scientists face using cloud services for scientific computing, including data organization as well as sharing and serving models [125]. Our interviews with AI/ML containerized app developers revealed how they could benefit from solutions such as DEEP, that provide more infrastructure

transparency whereby the underlying hardware, operating systems, and cluster management are abstracted away from the user. Indeed, our participants reported how they experience gulfs of evaluation from the tight integration between infrastructure-application knowledge (i.e., a full-stack understanding) that is required when using a Kubernetes cluster. This is especially true for evaluating dataset management tasks, as discussed in Chapter 5.2.2, with particular detail in Chapter 5.2.2.3. Our participants also reported experiencing gulfs of evaluation when performing job submission tasks due to the tight integration between infrastructure-application knowledge, as discussed in Chapter 5.4.2.

Garcia et al. note that data scientists face challenges in data organization due to the increasing availability of large amounts of data. These challenges include managing multiple tasks that may not be within a data scientist's job role expertise (i.e., provisioning the necessary computing resources, storage capacity, and orchestrating various infrastructure components in different places). The findings from our interviews support these challenges, as our participants reported that dataset management is one of the most challenging tasks in terms of the gulfs of execution and evaluation, in part, due to the increasingly large amount of data required to develop AI/ML containerized apps, as discussed in Chapter 5.2.1.1. In one of the case studies presented by Garcia et al., they found that the most time-consuming task when training ML models was gathering and cleaning the new datasets. This mirrors the reports heard in our interviews regarding how preprocessing datasets requires more computing resources and time than AI/ML model training, as discussed in Chapter 5.2.1.1.

Infrastructure-Application Knowledge.

Similar to the DEEP framework [125], Lee et al. built a Kubernetes ML development platform to support collaboration among developers with varying levels of ML and systems expertise, as they found that AI/ML containerized app development requires broader expertise in both ML and systems admin tasks [149]. Lee et al. provide their platform users with an IDE that includes the ability to ask for clarifying instructions and problem-solving help from other,

more experienced developers. Our interviews revealed how more integration is needed in AI/ML containerized app development, as developers use a multiplicity of tools, with little overlap. We also found that developers performing different job roles frequently communicate with one another, especially in order to accomplish tasks outside of their expertise.

Although Lee et al. described their collaboration feature in a case study presented for ML coursework projects, it is not clear for what tasks or at which point in the ML model development cycle the collaboration feature was used. Throughout Chapter 5, we discuss the specific AI/ML containerized app development tasks as well as the concepts that developers struggle with understanding and reach out for help with, or that lead to custom tools, educational materials, presentations, or more comprehensive documentation to be created. Furthermore, Table 6.1 provides quantitative data regarding these qualitative challenges. Therefore, we provide some of the missing context regarding the tasks and concepts that data scientists struggle to understand when employing Kubernetes for AI/ML containerized app development, which contributes to a clearer understanding of the challenges Lee et al. address in designing this platform.

As reported in our study, interfaces that consider developers with varying job role expertise, such as the OpenShift dashboard, provide developers with more relevant context for accomplishing their goals. For example, infrastructure details can be hidden by default in a novice developer interface and provide an option to switch views to see more detail. This would allow novices such as data scientist focus on developing AI/ML models and provide them with further insight when the experience gulfs of evaluation gulfs. For example, to address questions such as, “Why is my job running slower?” as discussed in Chapter 5.3.2.1.

Multi-Expert, Real-Time Coordination.

Several tool intervention studies address, in part, the challenges that AI/ML containerized app developers face due to increased communication with others for help. Below, we revisit these tool interventions, which include the: 1) Katib framework [127], 2) Scanflow-K8s framework [156], 3) OCDL platform [157] and, 4) Kubeflow service [129].

Regarding the first tool intervention, George et al. built Katib, an open-source Kubernetes-native framework to automate hyperparameter (HP) tuning, designed with a focus on two job roles (i.e., data scientists and operations engineers) that have different requirements and expectations from a HP tuning framework [127]. Operations engineers perform similar job roles to the cluster operators we interviewed, as shown in Tables 3.9, 4.8, and 4.9. Therefore, to the best of our understanding we correspond operations engineer to cluster operator, as shown in the third row of Table 6.2. Interestingly, our interviews revealed how the primary requirements of data scientists and cluster operators, as shown in Tables 3.9 and 4.8, overlap in practice. For example, requirements for operations engineers such as to “share the cluster with non-HP tuning jobs” and “identify issues in the system through logging and monitoring” are also required by data scientists. Therefore, although the separation of concerns regarding these different job roles is addressed by tools such as Katib, our interviews revealed that in practice, this is not the reality. The responsibilities of data scientists and cluster operators are not fully independent of one another and require developers to have a broader, overlapping set of expertise, at least in the organizations we interviewed. Therefore, as presented in their paper, the Katib framework would present barriers for data scientists and cluster operators to accomplish their work most effectively.

Our interviews with AI/ML containerized app developers, who perform HP tuning, revealed how it is one of several stages in the process of developing AI/ML foundation models, the others being training and inferencing. These tasks fall under the expertise of data scientists and come with their own challenges. Custom tools for ML model experimentation, such as the one introduced in Chapter 5.4.1.1, offer foundation models as a service via multiple modalities (e.g., APIs, SDKs, GUIs) to simplify access to large models such that AI/ML containerized app developers are able to independently experiment without needing access to costly computing resources or to communicate with a data scientist. A project manager overseeing the development of such a custom tool explains that although it has lowered the barrier for AI/ML containerized app developers to perform some tasks such as inferencing, it is still challenging to independently perform other tasks, such as HP tuning as this requires more expertise and a large amount of

their own labeled data, which act as significant barriers that widen the semantic and articulatory distance they experience:

(A19, Project Manager) *“[Using your own datasets within the custom ML model experimentation tool] is still difficult [even though] it’s enabled. You still need [data scientist] expertise. I wouldn’t say an application developer is going to go about tuning, you tend to need to be the data scientist and getting it right still requires expertise. We definitely lowered the barrier for tuning, not as dramatically as I think we did for inference.”*

As explained in Chapter 5.3.2.1, infrastructure developers often help others debug performance issues in large clusters as it requires understanding both application behavior and the underlying infrastructure. To debug performance issues, infrastructure developers create custom health check test tools to help identify the root cause of the issues and more quickly resolve them. An infrastructure developers emphasizes that debugging performance issues, similar to performing tasks such as HP tuning, are challenging as they must identify where bottlenecks exist in order to determine the root cause of the issues:

(A13, Infrastructure Developer) *“Performance debugging, I think it’s challenging for anyone. If you have worked performance debugging or tuning in the large cluster, you know how challenging it is to identify where the bottleneck [is]. Who knows where?”*

As discussed in Chapter 5, with particular detail in Chapters 5.2.1.1, 5.3.1, and 5.4.1.1, our interviews revealed that computing resource contention is one of the main challenges of performing dataset management, debugging, and job submission tasks in the context of AI/ML containerized app development. As well, as discussed in Chapter 5.2.1.1, job priorities impact how computing resources are allocated. A project manager explains that tasks such as HP tuning are “GPU-heavy” and require more resources, which are reserved on organization A’s Kubernetes cluster:

(A14, Project Manager) *“Sometimes just trying to get access to a cluster with the right resources [is difficult]. (...) As a team, [we] have been pretty lucky that we were able to get clusters spun up for us, which were quite large to do a lot of this [AI/ML containerized app development] work. But as we go more to the GPU-heavy workloads, with more AI stuff, there’s a push to using more common tools around the foundation model stack [such as organization A’s Kubernetes cluster]. It would probably be good if there was more coordination or [it was] more obvious [that] these are the clusters that you can use, this is how you can access [them], this is how you get something spun up. (...) There’s obviously [organization A’s Kubernetes cluster], which is big but its also very constrained. Those GPUs need to be used to train the big models, in an organized way, for doing things like fine-tuning and inference. A bit more organization and visibility around what’s available and how you get access and stuff (...) would be good.”*

The above quote also reveals how the lack of “organization and visibility” regarding what clusters are available to use and how to use them impacts productivity. The project manager quoted above emphasizes the need for better development processes to support developers in this regard. The Katib framework addresses cluster capacity planning by giving cluster operators the ability to define in advance computing resource limits for each developer, or experiment, to ensure that cluster resources are not oversubscribed. However, to our knowledge, Katib does not provide cluster users, or data scientists, with development processes regarding what resources are available and how to get access to them. Given that the Katib framework addresses computing resource contention issues, mirroring the reports heard in our interviews, perhaps we can expect to see that Katib users also experience similar gulfs of execution and evaluation. Our contributions towards understanding these challenges in terms of the gulfs as well as the distance that AI/ML containerized app developers experience can help further improve frameworks such as Katib.

Regarding the second tool intervention, Liu et al. built the Scanflow-K8s framework for MLOps, focusing on automating tasks that typically require collaboration between data scientists and cluster operators training and deploying ML models [156]. For example, they point out that after a team of data scientists performs repetitive AI/ML model training workflows, a different team of data engineers then deploy and manage the production-ready model. As a reminder, although they distinguish between data scientist and data engineer job roles, we do not report on data engineers as a separate job role in our study. Therefore, to the best of our understanding we correspond their data engineer job role to our cluster operator job role, as shown in the third row of Table 6.2.

In support of the challenges addressed, in part, by the Scanflow-K8s framework, our interviews also revealed that in the context of AI/ML containerized app development, developers across all job roles must perform tasks that span the entire development cycle, resulting in frequent collaboration from reaching out to others for help. This separation between infrastructure operations from application development was also found to be true in practice for general containerized development, from Zhou et al.'s findings regarding developers employing DevOps and microservices [211]. The automation offered by the Scanflow-K8s framework may help reduce some of this collaboration, although it is unclear to what extent. Conducting a user study to further understand how collaboration can be overall reduced through the automation provided by the Scanflow-K8s framework would help understand its benefits and identify any areas for improvement.

Furthermore, Liu et al. extended Scanflow-K8s to implement fine-grained scheduling for containerized HPC workloads in Kubernetes clusters [155]. Adapting jobs from a traditional HPC environment to a Kubernetes environment is a challenge that was frequently reported in our interviews, as discussed in Chapter 5, with particular detail in Chapters 5.2.2.1, 5.3.1.1, and 5.4.1.

Regarding the third tool intervention, Liu et al. built the open-source OCDL platform for MLOps and DevOps development to support collaboration among data scientists, project

managers, and application developers [157]. As a reminder, although they distinguish between application developers and other job roles, as shown in Table 3.12, we do not report on application developers as a separate job role in our study. Unfortunately, Liu et al. do not clearly explain the job role of application developers with the same level of detail used to explain data scientist and project manager job roles in Table 3.12. To the best of our understanding, we correspond their application developer job role to our data scientist, library developer, and infrastructure developer job roles, as shown in the fifth row of Table 6.2. This mirrors how we corresponded the application developer job role in the 2018 Sig Apps survey [120]. Therefore, we provide some of the missing context regarding the job roles and tools that these surveys report on, which contributes to a clearer understanding of the challenges Liu et al. address in designing their platform.

Liu et al. found that developers performing these different job roles all use their own tools along with different types of computing resources (i.e., CPUs, GPUs) to develop ML models, which results in a time-intensive collaboration process [157]. Their finding regarding how developers use a multiplicity of tools, with little overlap, mirrors our own findings and those in prior work on general containerized development [120]. Therefore, our conclusions about the gulfs and distance that developers experience from using a multiplicity of tools are supported in the context of both general containerized [120] and AI/ML containerized app development [157]. Similar to Liu et al., we also found that developers use different types of computing resources to develop AI/ML models, in part, due to computing resource contention. Liu et al. emphasize how using different tools and computing resources impacts collaboration, worsening the challenges experienced by AI/ML containerized app developers [157].

Regarding the fourth tool intervention, Golubovic and Rocha built a Kubeflow service to integrate ML development process management and supports collaboration among data scientists [129]. The challenges posed by a lack of integration as well as the use of a multiplicity of tools are mirrored in those addressed by other prior work on tool development, such as the platforms presented by Lee et al. [149] and Liu et al. [157], respectively. Indeed, our

interviews revealed how more integration is needed in AI/ML containerized app development, as developers use a multiplicity of tools, with little overlap. This mirrors findings in prior work on general containerized development [120]. Therefore, our conclusions about the gulfs and distance that developers experience from using a multiplicity of non-integrated tools are supported in the context of both general containerized [120] and AI/ML containerized app development [149, 157].

Golubovic and Rocha emphasize that by using an integrated service, similar workloads can be aggregated to provide a larger amount of resources to different teams and improve the efficiency of resource usage, which is especially important for AI/ML jobs that require scarce resources such as GPUs as well as workloads that fluctuate in resource demands [129]. Similar to the reports from our interviews with developers from two different size research organizations, computing resource contention is also an area of concern in other, large research organizations conducting large-scale AI/ML containerized app development, such as CERN.

Ineffective Experiment Tracking.

Molner et al. built SnapperML, an open-source Python framework for MLOps that provides data scientists with features that support reproducibility, including to efficiently track experiments and log hardware details [164]. Our interviews revealed similar challenges regarding reproducibility and experiment tracking in Chapter 5.4.2, with particular detail in Chapter 5.4.2.2. Furthermore, we found logging to be particularly challenging in the context of debugging tasks, as discussed in Chapter 5.3, with particular detail in Chapters 5.3.1.3 and 5.3.2.2.

Molner et al. designed the SnapperML framework to avoid code modification and reduce the possibility of introducing bugs or increasing technical debt [164]. In order to avoid code modification, they make use of YAML files due to their standardized structure, readability, and compatibility with version control software such as Git to ensure reproducibility. However, as discussed in Chapters 5.3.1, 5.4.1.4, and 5.4.1.5, this is not an ideal use for YAML files. As presented in their paper, it is not clear whether the SnapperML framework provides templates of

existing YAML. As the SnapperML framework is open-source, it's available for download via GitHub and we found that the repository does contain examples of YAMLs for running several different AI/ML experiments (e.g., SVM training) [163]. Importantly, such contradictions reveal the significance of conducting user studies with developers to understand how they, as users, would be benefited or challenged by the design choices made for such tool interventions. For example, a study with our participants using the SnapperML framework could help to uncover these contradictions.

In practice, our interviews revealed that, for developers with experience using a traditional HPC environment that are unfamiliar with Kubernetes, YAML file modifications are not a simple task and can require more time compared to those in a traditional HPC environment. Furthermore, we found that in organizations where YAML files contain secrets that cannot be version controlled using software such as Git, this can lead to challenges in collaboration, reproducibility, and configuration management, as explained in Chapter 5.4.1.5.

Although we only focus on three tasks in this dissertation, we also asked our participants about tasks that involve deployment logic, including writing YAML files and building images, as well as monitoring tasks. Developers across all job roles reported experiencing gulfs of execution in performing deployment logic tasks, as shown by the green bars in the middle column of Figure 5.1. Only data scientists reported experiencing gulfs of evaluation, as shown by the blue bar in the middle column of Figure 5.1. A data scientist explains that dealing with YAML files directly can be tricky for novices and takes their focus away from their main tasks:

(A22, Data Scientist) *“Generally, writing source code means writing some training files. So, we have a script (...) and add and modify as per the requirement. Then, we have our own YAMLs. These YAMLs take the parameters to schedule job for us. [We also have] YAMLs [concerned] with the cluster. (...) I generally don’t tamper around with those YAMLs. We have fixed those YAMLs so that novice users don’t have to worry that much about using [those] tricky things. [They] look very daunting for anyone. So, we generally try to keep [them] away from everyday users because otherwise [they’ll] get lost and won’t focus on [their] main job.”*

Molner et al. explains that one of the benefits of using YAML files is that the parameters allow for a limit to be set regarding the resources in a shared cluster environment. However, our interviews revealed that understanding how much of the available computing resources one needs to run their job comes with a learning curve, and is only part of the battle of computing resource contention. As explained in Chapter 5.4.1.1, unspoken “gentlemen’s agreements” regarding cluster use policies widen the semantic distance that developers experience. Not abiding by these policies widens the articulatory distance for other developers using the cluster, as they face further delays from the lack of available computing resources when the cluster is overloaded.

Our interviews further revealed that code modification results from developers switching between various cluster environments (e.g., a traditional HPC cluster and a Kubernetes cluster) in order to resume progress when they experience computing resource contention, as discussed in Chapter 5.4.1.3. As well, our participants reported that comparing logs or outputs across various clusters is difficult due to the lack of integration, complicating debugging and analysis. This highlights the need for better tooling integration to enable consistent workflows across diverse cluster environments. Therefore, our results contribute further understanding to the issues of code modification and logging, addressed by the SnapperML framework.

6.3.2.3 Takeaways on AI/ML Containerized App Development

As seen in these prior works, the AI/ML containerized app development community is reporting challenges that are echoed by the general containerized development community, including a divergence from specialized tasks due to the need for infrastructure-application knowledge (i.e., full-stack expertise) [149, 33, 13, 80] along with using a multiplicity of tools [80, 33, 13, 149, 129] and an increase in communication with others for help due to the need for multi-expert, real-time coordination [127, 156, 157, 129]. However, they also reveal challenges that are more unique to the domain of AI/ML, including dealing with large datasets [125] and ineffective experiment tracking [164]. As we conducted interviews with a similar community of containerized developers, perhaps we can expect to see that these survey respondents and the users of these tool interventions also experience similar gulfs of execution and evaluation when switching between AI/ML containerized app development tasks that may diverge from their expertise, using a multiplicity of tools. Again, the quantitative results contributed by the surveys provide a gauge for the significance of the qualitative challenges faced by the developers we interviewed. As well, perhaps we can expect to see that the users of these tool interventions experience similar gulfs when dealing with large datasets and tracking their AI/ML experiments.

These prior works on tool interventions focus on the perspectives of developers with specific job roles and some go into more detail defining the job roles or explaining the tasks performed within them. However, our interviews focus on even more developer perspectives, including: 1) cluster operators, 2) data scientists, 3) library developers, 4) infrastructure developers, and 5) project managers. In doing so, we found that the developers performing these job roles frequently communicate with one another as part of a larger development ecosystem. Furthermore, we define the job roles and tasks that AI/ML containerized app developers perform within as well as outside of their expertise, identifying the tools they use to perform these tasks. Therefore, we provide more context regarding the tasks that these developers perform within and outside of their job roles, which contributes to a clearer understanding of the challenges

addressed, in part, by these tool interventions. Understanding the tasks that developers perform outside of their jobs role may signal work that lacks modularity, indicating where developers may need the most help and motivating future improvements to such tool interventions.

Likewise, while these surveys and tool intervention studies report on the tools used by their participants, they do not report on the development task context in which tools are used, as we do. This is important to understand as, for example, even the case studies that George et al. briefly present regarding the Katib framework reflect how organizations use tools as part of a larger ecosystem [127]. Tool interventions could be further improved by understanding how they can be better integrated with the larger tool ecosystems that exist within organizations.

6.4 Recommendations for Future Work on Tool Interventions

Having discussed some of the implications of our results, there remain future opportunities. For each of the challenges shown in Table 5.1, we summarize the solutions presented by relevant work and make further recommendations for future work in tool intervention studies in Tables 6.3, 6.4, and 6.5. To save space in these tables, we omit the challenges shown in Table 5.1 for which our findings provide no further recommendations or for which future work does not address. In these tables, we also reference chapters and figures from this dissertation where tool interventions and further recommendations are discussed and shown in more detail regarding their limitations in addressing these challenges. Therefore, open areas for future work exist despite the relevant work cited.

Table 6.3. Summary of the specific challenges developers experience when performing dataset management tasks, whether these challenges are addressed in prior work on tool interventions, and recommendations for future work regarding development processes and tools based on our findings.

Dataset Management Gulfs	Challenges	Future and Relevant Work
Execution	Dealing with large datasets	Relevant Work: DEEP framework [125] (§ 3.3.2.2).
	Mounting storage and managing credentials using COS	Future Work: Tools . Custom MLOps platform to abstract away dataset management tasks (§ 5.4.2.2). Relevant Work: Dakic et al. [111] (§ 3.3.2.2). DEEP framework [125] (§ 3.3.2.2). Scanflow-K8s framework [156] (§ 3.3.2.2).
Evaluation	Adapting storage from HPC to Kubernetes	Future Work: Processes . Tracking metrics to reveal knowledge gaps (e.g., the time it takes a developer to successfully deploy their first job in a Kubernetes environment, the number of developers that are requesting help and the challenges they face). Relevant Work: Dakic et al. [111] (§ 3.3.2.2). Scanflow-K8s framework [156] (§ 3.3.2.2).
	Overcoming the learning curve for COS	Future Work: Tools . Custom MLOps platform to abstract away dataset management tasks (§ 5.4.2.2). Relevant Work: Dakic et al. [111] (§ 3.3.2.2). DEEP framework [125] (§ 3.3.2.2). Scanflow-K8s framework [156] (§ 3.3.2.2).
	Infrastructure-application knowledge	Future Work: Processes . Educational materials such as documentation and tutorials (§ 5.2.2.1, 5.2.2.2). Tools . Grafana dashboards (§ 5.3.2.1, 5.3.2.4, 5.4.1.1, 5.4.1.3, and Figures 5.6, 5.7). Relevant Work: DEEP framework [125] (§ 3.3.2.2).

Table 6.4. Summary of the specific challenges developers experience when performing debugging tasks, whether these challenges are addressed in prior work on tool interventions, and recommendations for future work regarding development processes and tools based on our findings.

Debugging Gulfs	Challenges	Future and Relevant Work	
		Relevant Work:	Scanflow-K8s framework [156] (§ 3.3.2.2).
Execution	Adapting debugging from interactive to batch sessions	<u>Future Work:</u>	<i>Processes.</i> Working up to running AI/ML jobs that require more resources and relying on checkpoints to resume progress (§ 5.2.1.1, 5.3.1.2, and Figures 5.6, 5.7). <i>Tools.</i> Jupyter notebooks to iteratively test AI/ML containerized app code (§ 5.3.1.2, 5.4.1.2, and Figures 5.10, 5.11).
	Overcoming cluster unreliability with checkpointing and iterative debugging	<u>Relevant Work:</u>	SnapperML framework [164] (§ 3.3.2.2).
	Manual process of logging	<u>Relevant Work:</u>	DEEP framework [125] (§ 3.3.2.2). Lee et al. platform [149] (§ 3.3.2.2).
	Multi-expert, real-time coordination	<u>Relevant Work:</u>	SnapperML framework [164] (§ 3.3.2.2).
Evaluation	Inadequate tooling for log analysis	<u>Future Work:</u>	<i>Tools.</i> Custom tools that perform automated health check tests to diagnose errors and automate debugging tasks (§ 5.3.2.1 and Figures 5.6 and 5.7).
	Limited visibility and uncertainty of an error's root cause		

Table 6.5. Summary of the specific challenges developers experience when performing job submission tasks, whether these challenges are addressed in prior work on tool interventions, and recommendations for future work regarding development processes and tools based on our findings.

Job Submission Gulfs	Challenges	Future and Relevant Work
Execution	Computing resource contention and manual coordination	<u>Future Work:</u> Processes. Publishing job priorities and expected computing resource needs in a centralized place, for all to see (§ 5.2.1.1, 5.3.1.1, 5.4.1.1, 5.4.1.3, 5.4.2.1). Tracking metrics to monitor computing resource contention (e.g., the time a developer must wait to use the computing resources they request, the time it takes for a developer’s job to finish running on the cluster). Tools. Custom ML model experimentation tool to simplify access to large models (§ 5.4.1.1 and Figures 5.10, 5.11).
	Users with varying experience and access	<u>Future Work:</u> Tools. Custom tools such as a job scheduler and a job submission CLI to simplify decisions for developers (§ 5.3.2.1, 5.4.1.3, and Figures 5.10, 5.11).
	Switching among multiple tools due to a lack of integration	<u>Relevant Work:</u> Katib framework [127] (§ 3.3.2.2).
	Infrastructure variability and hybrid cloud abstractions	<u>Relevant Work:</u> Golubovic and Rocha Kubeflow service [129] (§ 3.3.2.2).
	Configuring jobs using YAML	<u>Future Work:</u> Processes. Tracking metrics to reveal where more abstraction would be helpful (e.g., the time a developer spends performing tasks outside of their job role).
Evaluation	Managing unpredictable resource availability	<u>Future Work:</u> Tools. Bash scripts to automate the process of writing YAML (§ 5.4.1.3 and Figures 5.10, 5.11). Helm charts to automate the process of writing YAML (§ 5.3.2.1, 5.4.1.3, 5.4.1.5, 5.4.2.2, and Figures 5.10, 5.11).
	Ineffective experiment tracking	<u>Future Work:</u> Processes. Unspoken “gentlemen’s agreements” regarding cluster policies should be made explicit and public (§ 5.4.1.1). <u>Future Work:</u> Tools. Custom MLOps platform to support experiment tracking (§ 5.4.2.2). Tools such as Aim or MLflow (§ 5.3.1.3, 5.4.2.2, and Figures 5.10, 5.11). <u>Relevant Work:</u> Scanflow-K8s framework [156] (§ 3.3.2.2). SnapperML framework [164] (§ 3.3.2.2).

Throughout Chapter 5, we discussed how containerized developers experience semantic distance while learning how to perform unfamiliar tasks such as dataset management using COS, along with other DevOps tasks. Furthermore, we discussed how these gulfs reveal that containerized developers experience articulatory distance from having to spend time communicating with other experts whenever they are unable to independently accomplish tasks such as dataset management, debugging, and job submission.

As explained in Chapter 4.1.3, the gulfs can be addressed by reducing the semantic distance experienced on several fronts to increase direct engagement, including: 1) on the system side, requiring effort on the part of the system designer, 2) on the user side, requiring effort on the part of the user, or 3) via a combination of these perspectives, users can also act as system designers by creating their own custom tools, which work in continuity with how they already think. As a reminder, in our study, we assume that our participants are using well-designed tools. Therefore, our recommendations focus on the user side (i.e., solutions 2 and 3).

6.4.1 Solutions From the User Side

Importantly, our interviews revealed how there is a limited number of experts with sufficient infrastructure-application knowledge (i.e., full-stack expertise) and that this limited number is shrinking, not growing, as explained in Section 5.4.1.2. This finding mirrors the wider containerized development community, as discussed in Chapters 3.3.1.1 and 6.3.1.1. Having a full-stack expertise is an unrealistic expectation of a developer that is trained to perform specialized tasks within their job role and results in them experiencing gulfs of execution and evaluation. Without a full-stack expertise, tasks such as dataset management, debugging, and job submission are more difficult to independently accomplish, which signals the semantic distance that developers experience.

Many of the developers we interviewed reported that they too lack this full-stack understanding such that they become stuck and must reach out for help from the few who have such knowledge. This further signals the articulatory distance that developers experience, as they

wait for help from others in order to resume progress. As there are fewer experts available to help, better development processes and tools are needed to help facilitate knowledge sharing and alleviate the burden that those few experts currently bear. On the user side, such processes involve creating educational materials such as documentation and tutorials to share with others, establishing a foundation of knowledge and providing helpful references for novices. For example, providing a project- or team-specific GitHub Issues or a Wiki could help to facilitate answers to questions as a form of living documentation. A cluster operator explains that having a more integrated source for helpful references could be an improvement to current, "painful" development processes that have participants searching among many different places (i.e., Slack, GitHub, and Box):

P48, Cluster Operator) "Some people only put [documentation] in the Slack [channel] only, or on the private Slack channel only. (...) Also there [is] some [documentation] in the GitHub Issues or Wiki, some are in that Box Note. A lot of information is distributed (...) or hiding and [it is not] easy to [find] everything. [It is] not very easy and not very integrated, that is a pain. Sometimes [the knowledge has not been shared and] we need to [ask the someone] directly (...) what is going or how to do that. (...) We should do more, expand or improve the stuff there. (...) [We] should prepare some pathway or simple pipeline to keep everything in GitHub Issues, do not use [multiple sources] (...) for information sharing."

Such a form of living documentation must be built up over time and will require experts to invest their effort into providing the answers, at least initially. If experts invest their time into writing such documentation, it could reduce the interruptions they experience from novice developers asking for their help. As well, given the existence of generative AI tools such as ChatGPT [169], Claude [78], and Gemini [144], the task of finding all of the existing documentation and compiling it into a single hub could be made easier by providing experts with a starting point to further flesh out. Later on, the burden of maintaining such documentation

not be on the experts, as the more common questions will have been answered and will serve to provide helpful references for others in the future. As well, maintaining such a project- or team-specific SoF via GitHub Issues or a Wiki could help to more quickly reveal the challenges that developers experience and provide feedback on what improvements, regarding the development processes and tools used on the user or system side, are needed in the community.

Some of our recommendations involve the tracking of metrics, which is a standard practice in, for example, Agile software development processes [175]. In software development organizations, such metrics can be automatically tracked by GitHub. For example, a team's development velocity is continually monitored by the time it takes to complete tasks. Furthermore, discussions around issues happen in a centralized space where developers can assign one another to review code commits and issues, discussing any changes and solutions there in detail. Indeed, in Chapter 3.2.1.1 we discussed prior work on a large-scale analysis of a GitHub repository that made use of this information to understand the challenges developers face [210].

However, perhaps these Agile practices have not been fully adopted by our participants yet. For example, many of the data scientists that we interviewed expressed that although they develop code, they do not have formal training in software development. On the user side, these practices should be implemented by the organizations we interviewed to help them understand and continually monitor the challenges they face in developing AI/ML containerized apps.

6.4.2 Solutions From the System Side

On the system side, such solutions involve custom tools, for example, to determine the root cause of an error, as explained in Section 5.3.2.1. While such custom tools may help bridge the semantic distance that containerized developers experience from requiring such a full-stack understanding, the articulatory distance that they experience may still remain. In this case, the articulatory distance depends on the amount of communication a user must have with other teams regarding how to resolve the root cause of an error.

To reduce the semantic distance experienced on the system side, tool designers could

focus on abstracting away the tasks outside of a developer's expertise, that they struggle with, as described by the library developer quoted earlier. However, by abstracting away these tasks, such custom tools must still be able to support a developer's learning curve such that they are able to more independently execute and evaluate their tasks. For example, the OpenShift web console dashboard provides two different views in their GUI: 1) admin and 2) developer, to reflect the level of cluster privileges that a user has and provide different metrics for monitoring [137]. The idea to provide two different views reflects how such tools can be built to support developers as they gain expertise, from being a user or developer of the cluster, to being an admin.

Reducing the articulatory distance experienced is more reliant on the system side. For example, on the input side, providing an interface that permits the specification of an action by mimicking it (e.g., memorable relationships between the forms and the meanings of command names used in a CLI). For example, similar to the feature provided in Lee et al.'s platform, collaboration tools can be better integrated to give more context and reduce the cognitive load that developers experience from switching between tool contexts [149]. Furthermore, on the output side, providing a live, graphical display to continuously evaluate the status of an action that has been executed (e.g., a progress bar that reflects the amount of time left for completing an action that has been executed in a GUI). Although some of the challenges identified in our study have been addressed on the system side by prior work on tool interventions, they have limitations, as discussed in the chapters referenced throughout Tables 6.3, 6.4, and 6.5. Therefore, open areas for future work in development processes and tools exist.

Chapter 7

Threats to Validity

7.0.1 Internal Threats to Qualitative Studies

We address two threats to internal validity. First, due to the qualitative nature of our study, we may have made incorrect inferences when coding and analyzing our dataset of developer interviews. Therefore, we may have wrongly interpreted people's answers. As we conducted interviews with people, they may have misrecalled or exaggerated in their answers. As well, they may have wanted to please the interviewer, anticipating answers that they thought were desired by the interviewer.

Regarding any incorrect inferences we may have made when coding and analyzing our dataset, Chapters 4.5, and 4.5.1 discuss our data collection, coding processes, and limitations we faced in generating an ICA/ICR metric for our codebook, respectively. Our coding process was mostly straightforward (e.g., job role, experience) and required some level of interpretation (e.g., gulfs of execution and evaluation). However, we faced some limitations given that two of the researchers collaborating on our study could identify one or more participants based on their statements even after anonymizing the transcripts due to close collaborations or other conflicts of interest. Therefore, they recused themselves from individually coding the data. As only I coded the data, it was a longer process to complete than if we had multiple people coding the data. However, there are the risks that come with having multiple coders, such as maintaining a consistent form of agreement between them. Therefore, in having a single coder, we sacrificed

the speed in our coding process in order to maintain consistency, and thus a form of agreement.

In qualitative studies, subjective judgments are made. To eliminate any errors in these judgments, currently the best practices involve discussing and reviewing them further with others. Methodologies such as negotiated agreements and audits exist, having been used in prior work to help eliminate such errors [154, 126]. However, eliminating these errors only means that researchers have reached some level of agreement with one another (i.e., some level of reliability), not that their judgments are true or guarantee validity [130]. Following the recommendations by Sedano et al. and Campbell et al. [189, 100], I met regularly with my collaborators to go over my coding progress. During these meetings, we would review any new or changed codes, discussing them to reach an agreement on their interpretation. This iterative coding process resulted in a final codebook.

Regarding participants misrecollecting or exaggerating in their answers, our results are reported in a manner following Cito et al. [104], such that our findings are supported by the statements of multiple participants (i.e., at least two). Related to this, participants may not have felt comfortable to speak openly about their challenges during the interviews. Therefore, I anonymized the data and conducted the interviews in a private setting where the discussion could not be observed or overheard by others. The interviews were conducted either in-person, in the familiar context of a participant's office or a meeting room on-site of the participant's work place, or remotely via Webex or Zoom.

Second, most of the researchers collaborating to conduct our study worked with our participants or held higher positions than the participants within the organizations we interviewed. Therefore, in an effort to conduct the interviews without added bias and social pressure, as a neutral third party to both organizations, I conducted the interviews. I did not hold higher positions or have educational degrees that were more advanced than the participants'. Given that myself, a junior researcher conducted the interviews, our participants may not have felt as much pressure to please the interviewer.

Another concern with many studies is the external validity of the results. To minimize

this threat, we interviewed professional software developers in their work settings. However, there are still questions on how these results relate to other academic and industrial settings as well as other areas in the domain of AI/ML.

7.0.2 Limits to Generalizability

We address four limits to generalizability. First, we interviewed participants from research organizations, as opposed to traditional software development organizations. However, participants from organization A were developing products for clients, such as AI/ML model apps, in an industry research and development setting. It's possible that other traditional product-oriented organizations have different mappings between a job role and the tasks expected of that role than those we encountered. Interviewing more AI/ML developers in different workplaces (e.g., research, industry) to explore whether our findings can be generalized, is a point of consideration for future work. Related to this, as we focus specifically on AI/ML containerized app developers, we would need to conduct more interviews with non-AI/ML containerized app developers to further understand whether our results generalize.

Second, as technology continues to change and progress at a rapid pace, AI/ML workloads are rapidly evolving with it. Around the time that we conducted our interviews, the emphasis of AI/ML workloads was on data preparation and large-scale AI/ML model training [13]. Now, workloads focus on large-scale AI/ML model fine-tuning and container orchestration of heterogeneous AI/ML models [33]. A point of consideration for future work would be to help discern how job role-task mappings are evolving along with these AI/ML workloads. Although the emphasis has somewhat shifted, data preparation and large-scale AI/ML model training are still relevant, impacted by the scale at which AI/ML workloads have grown. For example, as the datasets get larger, they may introduce more privacy-related issues.

Third, Kubernetes itself is rapidly evolving. Originally, Kubernetes came about as a way to structure the use of containers for hosting cloud service apps [97]. More recently, the Kubernetes community has realized the importance of high performance computing (HPC)

workloads. For example, similar to SIGs such as SIG Apps, other working groups (WGs) such as the Batch Working Group strive to improve support for batch (e.g., HPC, AI/ML, data analytics, CI) job scheduling workloads in Kubernetes [146]. As part of that effort, the Kueue project offers improvements to the way jobs are queued [147]. Such improvements would likely help address some of the gulf experienced by our participants.

As the gulfs may be narrowing due to improved technology, tools, and processes they may also be widening with the increasing size of datasets and the complexity of techniques this scale introduces. Issues of scale and complexity generate increased inter-job role communication as well as lead them to perform multiple tasks, experiencing additional challenges, as discussed in Chapters 5 and 6. Especially in fast-paced fields such as technology, the gulfs framework provides support for understanding these challenges.

Related to how technology, AI/ML workloads, and Kubernetes itself are rapidly evolving, we conducted our interviews during the summer of 2023. We must acknowledge that given several years have since passed, things may have changed for our participants. Despite this, AI/ML containerized app development is still a relatively new use for Kubernetes, as discussed in Chapters 3.3.1.2 and 3.3.2.1. As well, we found that our results are still relevant to the findings of even the latest CNCF survey conducted in 2025, as discussed in Chapter 3.3.2.1. However, a point of consideration for future work would be to revisit these organizations to conduct follow-up interviews with our participants to understand how much has changed for them in the time since their initial interview.

Fourth, there is a tool usage gap. Even at the time of our study, tools existed that could have helped address some of the gulfs experienced by our participants. In part, due to a combination of a lack of awareness and the intentional dismissal of tools due to team inertia, tools that may have helped were not employed by these teams. For example, TorchX helps with submitting jobs to a Kubernetes cluster by building and publishing container images on a user's behalf [202]. Although TorchX was known by some of our participants, they reported that they intentionally chose a different path and did not use it. A point of consideration for future work

would be to understand the impact of this tool usage gap.

Chapter 8

Conclusion

Containers propose benefits to software development and scientific research projects, such as improving reproducibility and collaboration among teams who may use different tool ecosystems [165]. Despite these purported benefits, **research had yet to provide a deeper understanding of the challenges faced by those employing containerized development in domains for which it wasn't originally anticipated**, notably AI/ML. This is especially relevant as AI/ML workloads continue to change and progress at a rapid pace, introducing more complexity in containerized development.

Anecdotally, we had heard from larger teams of software developers that employing Kubernetes for AI/ML containerized apps has proved to be challenging. As discussed in Chapter 4.1.1, larger teams often have distinct job roles with specialized task responsibilities that are intrinsic to those roles. Therefore, we hypothesized that **AI/ML containerized app development, even as orchestrated through Kubernetes, was undermining the separation of responsibilities among roles**, impacting a team's modularity of work.

To give us a sense as to the nature of how the separation of responsibilities may be undermined or to what extent this may impact a developer's work, we applied the gulfs of execution and evaluation as an analysis framework, in a top-down fashion. Furthermore, although the **theories of gulfs and specialized labor** provide a powerful analytical lens, we could not presume to know, in advance, how our participants' narratives, experiences, and work would

or wouldn't either flesh out or fit into these concepts. Therefore, we followed the techniques of **Straussian Grounded Theory**, the analysis framework most commonly applied by interview studies that focus on software developers [116, 211, 187, 189, 205, 145]. Using this technique, we coded the interview data to develop new codes beyond the two existing gulfs codes, in a bottom-up fashion.

Importantly, there was a **missing piece among prior work on understanding the challenges faced by developers employing Kubernetes for AI/ML containerized apps**. Given that the focus of the Kubernetes tool interventions discussed in Chapter 3.3.2.2 was to build a solution, to our knowledge, none of them conducted any in-depth user studies to understand how their tools are used in practice and further support their solution. To fill this gap, **we conducted interviews, contributing more depth to these prior works** by providing qualitative data to help further contextualize and explain the reasons behind the challenges faced by developers employing Kubernetes for AI/ML containerized app development. Furthermore, we focused on finer-grained job roles (i.e., data scientists, cluster operators, infrastructure developers, library developers, and project managers) and tasks (i.e., dataset management, debugging, image building, monitoring, deployment logic, job submission, cluster observation, application source code, and application versioning) than prior work reports on.

8.1 Takeaways

Throughout this dissertation, we answered the following **research question: As people in larger teams often have distinct job roles with specialized task responsibilities that are intrinsic to those roles, to what extent does AI/ML containerized app development impact the separation of responsibilities among roles?** Our interviews provided further insight to understand what tasks these developers are actually performing, despite their job role, and what's really troubling them, in practice. Below, we review the five main takeaways from the results of our study.

Takeaway 1: The knowledge expertise and computing resources required to perform job submission, debugging, and dataset management tasks in a shared cluster environment create challenges for AI/ML containerized app developers.

Figure 5.1 shows that overall, developers experience a high level of non-modular work, such that, with few exceptions, everyone is doing a bit of everything. We identified the **top three tasks that developers experience gulfs of execution and evaluation**, which include: **1) job submission, 2) debugging, and 3) dataset management**, as discussed in Chapters 5.4, 5.3, and 5.2, respectively. We also conducted a thematic analysis of the challenges found within these tasks and identified the extent to which different problems are experienced, as shown in Table 6.1 and discussed in Chapter 6.1.

Together, the gulfs in Figure 5.1 and the extent of the challenges presented by these gulfs in Table 6.1 support our hypothesis that **AI/ML containerized app development undermines team modularity**, revealing three repeated patterns across dataset management, debugging, and job submission tasks: **1) a divergence from specialized tasks, 2) switching between a multiplicity of tools, and 3) increased communication with others for help**, increasing the distance that developers experience.

Takeaway 2: Although only a few our of participants have a full-stack understanding, we found that most every one of them are doing a bit of everything, performing tasks outside of their job role expertise with added friction.

Regarding the first pattern, **developers reported that they frequently perform tasks that are beyond their expertise**, as discussed in Chapter 5, with particular detail in Chapters 5.2.2.3, 5.3.2.1, 5.3.2.3, 5.4.1.2, 5.4.1.4, and 5.4.2.1. Figure 5.1 supports this, showing that, with few exceptions, developers that are not trained with full-stack expertise, must perform as such. This pattern feeds into the other two we identified, as developers address the challenges of performing tasks beyond their expertise in three ways, including: 1) on the system side, 2) on the user side, and 3) using a combination of the two. Chapter 4.1.3 introduces these three perspectives along with their tradeoffs.

Takeaway 3: Switching between a multiplicity of tools is required to perform AI/ML containerized app development, adding complexity to workflows and increasing a developer's cognitive load.

Regarding the second pattern, **developers reported that they make use of a multiplicity of tools, including both off-the-shelf and custom-made, to help bridge the gulfs they experience**, as discussed in Chapter 5, with particular detail in Chapters 5.2.1.1, 5.2.2.1, 5.3.1.2, 5.3.1.3, 5.3.2.1, 5.3.2.3, 5.4.1.1, 5.4.1.3, 5.4.1.4, and 5.4.2.2. Furthermore, Chapters 5.2.3, 5.3.3, and 5.4.3, reveal that, across all job roles, developers on average use multiple tools for dataset management, debugging, and job submission tasks, respectively. Therefore, everyone bears the burden of learning how to use and switch between multiple tools to perform these three tasks.

Takeaway 4: Without a full-stack understanding, developers cannot work independently and must reach out for help, further slowing down high-stakes AI/ML containerized app development that is already constrained by shared resources and under tight deadlines.

Regarding the third pattern, **developers frequently reported that they reach out for help to resolve issues**, as discussed in Chapter 5, with particular detail in Chapters 5.2.2.3, 6.1.1, 5.3.2.1, 5.3.2.3, 5.4.1.2, and 5.4.1.1. More specifically, Chapter 5.3.3 reveals that the most used tool for debugging is a communication tool that developers use to message others for help. The third and fourth most used tools also regard means for communicating with others for help. The extent to which developers reach out for help further reveals a lack of modularity in their development process, as independent work becomes impossible and additional communication becomes necessary.

Takeaway 5: AI/ML containerized app development introduces unique challenges that involve experiment tracking and dealing with large-scale data.

As well, in Chapter 6.3 we discussed the **extent to which the challenges we identified are unique to containerized development or AI/ML**. This is especially relevant as we found that many of our participants conducted AI/ML containerized app development in a traditional HPC environment prior to employing Kubernetes. Overall, we found that both general and

AI/ML containerized app development present challenges that involve the three patterns we identified. However, the finer-grained challenges we identified, such as those that deal with **adapting between HPC and Kubernetes environments** are only reflected in prior work on general containerized development and have yet to be considered more specifically in the context of AI/ML. Furthermore, we identified the challenges that are unique to AI/ML containerized app development, such as **experiment tracking and dealing with large-scale datasets**.

8.2 Contributions

***Contribution 1:** We provided our interview questions and codebook to enable others to better evaluate as well as more easily replicate our qualitative study.*

One of the main contributions of our study are the **contextual interviews that we conducted with AI/ML containerized app developers**, as research had yet to understand their perspective as early adopters of Kubernetes in the domain of AI/ML. The questions we asked to our participants during these interviews are provided in Appedix A. We also provided our final codebook in Appedix C to share the codes and their definitions, developed throughout our coding process. Although we do not include our coded interview dataset in full, including these materials from our study makes them publicly available for others to evaluate and potentially use or build on in future work.

***Contribution 2:** We demonstrated how combining several, well-known analytical frameworks can be used to identify and more deeply understand the challenges AI/ML containerized app developers face, fueling prior and future work.*

The other main contributions of our study include how **we employed a combination of bottom-up and top-down analysis frameworks to analyze our interview data**. The bottom-up aspect of this dual analytical lens allowed us to conduct more exploratory research, employing Straussian Grounded Theory to code our interview data. The top-down aspect of this dual analytical lens allowed us to apply the gulfs of execution and evaluation to understand AI/ML containerized app development. As well, **we aligned the gulfs with the idea of work partitioning according to specialization** to further understand AI/ML containerized app development in

the context of a developer's job role, the tasks they perform, and the tools they employ.

By employing this dual analytical lens, we were able to understand how the **tasks that developers perform outside of their jobs roles may signal work that lacks modularity**, indicating where developers may need the most help and motivating future improvements to tool interventions. In Chapters 6.3.2.2 and 6.4, we addressed what parts of the **tool interventions presented in prior work are promising for AI/ML containerized app development** and how they could be further improved. We find it encouraging that progress has been made to address some of the challenges that were also reported in our interviews.

The unresolved challenges we identified are helpful for both system designers and users to consider when employing Kubernetes for AI/ML containerized app development. Further classifying these challenges into the gulfs of execution or evaluation narrows down **on which front more effort is needed**, including: **1) on the system side, 2) on the user side, and 3) using a combination of the two**. Our interviews with AI/ML containerized app developers contribute a clearer understanding of the **tradeoffs involved in bridging these gulfs** from these three perspectives. These unresolved challenges offer open areas for future work, providing **points of caution for common challenges and starting points for potential improvements** to guide others in their endeavor to employ containers for AI/ML app development. Understanding a team's modularity of work is crucial to addressing the challenges they face and maintaining their productivity.

Appendix A

Interview Questions

A.1 Warm-up

First, I'd like to start with getting to know you a bit more and asking some background questions:

1. What are your job roles (e.g., cluster operator, data scientist, infrastructure developer/resource manager, library developer, project manager, UX designer/front-end developer)?
2. How many years of professional software development experience do you have?
3. Could you talk about how you use Kubernetes (e.g., when did you begin to use Kubernetes, when did you begin to finally feel comfortable using Kubernetes)?
4. What were you using to run AI/ML jobs before using Kubernetes (e.g., HPC platforms)?
5. How frequently do you work on applications using Kubernetes (e.g., never, monthly or even more rarely, weekly, once a day, many times each day)?
6. In general, how many different apps are you actively contributing to (e.g., could you talk about how many different apps you typically work on every day)?
7. Could you show me an example of an app you've developed or are actively working on (e.g., what does it look like, how do you use it, what is the source language, what is the goal of the app)?

A.2 Everyday Process

Next, I'd like to ask you about your everyday process when working on these apps:

8. When working on a single app or multiple apps, what kinds of things are you doing (e.g., how frequently are you switching between these things, how urgent are these things)? Specifically, how much time do you spend:

- Authoring AI/ML source code (e.g., Torch, Ray)
- Authoring or editing YAML (e.g., forms of deployment logic)
- Curating and managing datasets
- Building images (e.g., managing dependencies)
- Arranging a job for submission (e.g., sizing the run, coordinating for time on the cluster)
- Monitoring the progress of a run
- Observing cluster-wide health (e.g., utilization logs/metrics)
- Debugging an AI/ML run to determine why its misbehaving

9. If you take any notes as part of your development process, what do you use them for? Specifically, do you use notes to help:

- Refer back to things (e.g., documentation, helpful code examples)
- Keep track of errors and solutions (e.g., current implementation challenges)
- Keep track of ideas (e.g., ways of improving things, new ideas)

10. Could you show me an example of your notes?

A.3 Development Process

We're about halfway through the interview and next I'd like to ask you some more detailed questions to learn more about your development process in four ways:

11. Regarding onboarding, when you have a new team member, how do you get them up to speed so that they can run AI/ML jobs by themselves (e.g., process, tools, differences using Kubernetes versus HPC platform, documentation)?
12. Regarding app versioning related to code dependencies, when you create a new app or a new version of the app, how do you propagate it, how do you test it, is the release process automated or manual, do you need to build anything to make the release happen, or just using git (e.g., process, tools, differences using Kubernetes versus HPC platform, documentation)?
13. Regarding datasets, when you create a new dataset, is there a place that you store it or does the new dataset need to be approved by anyone (e.g., process, tools, differences using Kubernetes versus HPC platform, documentation)?
14. Regarding the mechanics of submitting an AI/ML job, how do you accomplish that (e.g., process, tools, differences using Kubernetes versus HPC platform, documentation/history for personal use or team use like Slack)?
15. Pertaining to questions 11–14, regarding tools:
 - Do you build tools for yourself or others (e.g., scripts), are they common to your project (e.g., do you use tools that others on your team built, what was your experience using the tool, do you feel that you're able to use it effectively), or are they generic COTS software?
16. Pertaining to questions 11–14, regarding differences:

- You mentioned earlier that before using Kubernetes to run AI/ML jobs, you used X (e.g., Slurm, Spark), how is developing using Docker or Kubernetes different from developing using X (e.g., an HPC platform)?

17. Pertaining to questions 11–14, regarding documentation:

- What/who is it for (e.g., getting started guides, tutorials, quick-starts, summarizing results)?
- Can you show me an example of your documentation (e.g., what modality do you use for documentation, documentation-as-code, Markdown or Jupyter, Slack, Box)?

A.4 Tools

We're about three-quarters of the way through the interview and next I'd like to learn more about the kinds of tools you use for your everyday work:

18. How much time do you spend building tools for yourself or others (e.g., one-off cost, constant)?
19. How much time do you spend learning new concepts or how to use new Kubernetes tools (e.g., one-off cost, constant)?
20. What tools do you use to complete your daily work? Specifically, what tools do you use to help with crafting, running, and observing AI/ML jobs:
- Writing code for AI/ML tasks (e.g., writing Python code)
 - Writing code to deploy AI/ML tasks to a cluster (e.g., writing YAML code or Helm charts)
 - Curating and managing datasets
 - Building images (e.g., building Docker or VM images)

- Sizing, planning, and submitting jobs to a cluster
- Observing a job run (e.g., monitoring for progress)
- Observing a cluster (e.g., monitoring for infrastructure/management roles)
- Debugging

21. Are there any other tools you use to complete your everyday work? Specifically, do you use a web-browser to:

- Search for documentation, helpful code examples, or solutions on Stack Overflow
- Learn about new concepts in Docker or Kubernetes
- Learn about how to use tools for Docker or Kubernetes

22. What's it like using these tools all together? Specifically, are these tools:

- Well-integrated into your development environment

A.5 Typical Scenario

Finally, I'd like to ask one more question to learn about what a typical day is like for you using Kubernetes (e.g., the kinds of tasks you regularly complete, any issues you needed to resolve to complete them):

23. Perhaps recalling an example of what your day was like yesterday, or another time if yesterday wasn't a typical day. You get into the office, open your laptop, you probably have to get your `oc` login key, then what do you do (e.g., are you cloning a new repo using `git clone`, updating your code using `git pull`, checking the status of a run in the OpenShift console, reading Slack updates from your teammates, does it all go smoothly, at what point do you become frustrated)? Specifically, take me through the steps of resolving an issue you frequently run into in your everyday work and show me any relevant examples of:

- Information you used (and in what order) to identify the root cause of the issue
- Tools used for generating/displaying information
- When it becomes necessary to use multiple tools (e.g., how often you switch between tools)
- How you keep information straight as you switch between tools
- How you combine these tools (e.g., arrange them on displays)
- If code was involved and you were using version control, the “before” and “after” code
- How you get un-stuck (e.g., ask teammates for help in-person or via Slack, post on Stack Overflow, submit an issue on GitHub, invent a new tool)

A.6 Wrap-up

To wrap-up, I just have two more questions to get your feedback on this interview:

24. Is there anything else you’d like to say or share today with respect to using Kubernetes?
25. Do you have any comments or feedback about the questions I asked you today?

Appendix B

Development Tools

B.1 Dataset Management Tools

Tables B.1, B.2, B.3, B.4, and B.5 further break down the tools that participants use for dataset management tasks by job role. These tools include both off-the-shelf and custom tools, invented by the participants themselves and indicated by the word “Custom” preceding the name of the tool in parenthesis. Communication tools are indicated by the word “Communication” preceding the name of the tool in parenthesis, or more generally using the format of communication (i.e., in a meeting or otherwise in a 1-1 interaction).

Table B.1. 13 unique tools are used by cluster operators to perform dataset management tasks.

Dataset Management Tool	Total (Number of Cluster Operators)
Nextcloud	2
S3	2
Central Ceph	1
Ceph	1
Dask	1
Eastern Ceph	1
LINSTOR	1
Open Science Grid Origins	1
Pandas	1
RAPIDS	1
Ray	1
seaweedfs	1
South Eastern Ceph	1

Table B.2. 11 unique tools are used by data scientists to perform dataset management tasks.

Dataset Management Tool	Total (Number of Data Scientists)
COS	10
Checkpoints	5
Custom (MLOps platform)	2
Hugging Face	4
GPFS	2
Bluepile	1
CVAT	1
Dask	1
NFS	1
PyArrow	1
Red Hat Open Data Hub	1
XArray	1

Table B.3. 10 unique tools are used by infrastructure developers to perform dataset management tasks.

Dataset Management Tool	Total (Number of Infrastructure Developers)
COS	4
S3	3
CubeFS	1
Checkpoints	1
EFS	1
FTP	1
IBM Cloud	1
Nextcloud	1
QGIS	1
seaweedfs	1

Table B.4. Four unique tools are used by library developers to perform dataset management tasks.

Dataset Management Tool	Total (Number of Library Developers)
AWS	1
COS	1
Checkpoints	1
Hugging Face	1

Table B.5. Nine unique tools are used by project managers to perform dataset management tasks.

Dataset Management Tool	Total (Number of Project Managers)
COS	3
S3	3
AWS	1
Checkpoints	1
GitHub/GitLab	1
Hugging Face	1
IBM Cloud	1
ImageNet	1
Mongo	1

B.2 Debugging Tools

Tables B.6, B.7, B.8, B.9, and B.10 further break down the tools that participants use for debugging tasks by job role.

Table B.6. 12 unique tools are used by cluster operators to perform debugging tasks.

Debugging Tool	Total (Number of Cluster Operators)
Communication (Matrix)	2
Communication (Slack)	2
Logs	2
Communication (1-on-1)	1
Communication (Meetings)	1
FlameGraph	1
GitHub Copilot	1
less	1
oc	1
SED	1
strace	1
TensorBoard	1

Table B.7. 15 unique tools are used by data scientists to perform debugging tasks.

Debugging Tool	Total (Number of Data Scientists)
Logs	13
Communication (Slack)	9
Communication (Meetings)	4
VS Code	3
Aim	2
Custom (Scripts)	2
TensorBoard	2
Chrome Tracing	1
Communication (1-on-1)	1
Custom (Documentation)	1
Jupyter Notebook	1
Matrix	1
OpenShift	1
Grafana	1
Travis CI	1

Table B.8. 20 unique tools are used by infrastructure developers to perform debugging tasks.

Debugging Tool	Total (Number of Infrastructure Developers)
Communication (Slack)	6
Communication (1-on-1)	5
Logs	5
kubectl	4
Communication (Meetings)	3
oc	3
OpenShift	3
VS Code	3
Argo	2
Custom (Automated health check tests)	2
Grafana	2
Jupyter Notebook	2
Matrix	2
Custom (Documentation)	1
k9s	1
Knative Eventing	1
Netcat	1
tcpdump	1
Unreal Engine	1
Wireshark	1

Table B.9. Two unique tools are used by library developers to perform debugging tasks.

Debugging Tool	Total (Number of Library Developers)
Communication (Slack)	2
Communication (Meetings)	1

Table B.10. 12 unique tools are used by project managers to perform debugging tasks.

Debugging Tool	Total (Number of Project Managers)
Communication (Slack)	4
Communication (Meetings)	3
Communication (1-on-1)	2
Logs	2
OpenShift	2
Cyberduck	1
GitLab	1
k9s	1
Communication (Matrix)	1
PyCharm	1
Vim	1
VS Code	1

B.3 Job Submission Tools

Tables B.11, B.12, B.13, B.14, and B.15 further break down the tools that participants use for job submission tasks by job role.

Table B.11. Six unique tools are used by cluster operators to perform job submission tasks.

Job Submission Tool	Total (Number of Cluster Operators)
Custom (Web-based remote desktop)	1
Helm	1
Jupyter Notebook	1
k9s	1
Lens	1
PyTorch	1

Table B.12. 14 unique tools are used by data scientists to perform job submission tasks.

Job Submission Tool	Total (Number of Data Scientists)
Helm	8
Custom (Script)	5
oc	4
PyTorch	3
Custom (Job submission CLI)	2
Custom (MLOps platform)	2
DeepSpeed	2
Spark	2
bsub	1
Jupyter Notebook	1
kubectrl	1
Ray	1
TorchX	1
VS Code	1

Table B.13. 12 unique tools are used by infrastructure developers to perform job submission tasks.

Job Submission Tool	Total (Number of Infrastructure Developers)
oc	7
kubectl	4
Custom (Script)	3
Helm	3
Ray	3
Spark	3
VS Code	3
Custom (Job submission CLI)	2
Jupyter Notebook	2
k9s	1
PyTorch	1
TorchX	1

Table B.14. Seven unique tools are used by library developers to perform job submission tasks.

Job Submission Tool	Total (Number of Library Developers)
Custom (Script)	2
Custom (Job submission CLI)	1
Jupyter Notebook	1
oc	1
Ray	1
Spark	1
TorchX	1

Table B.15. 14 unique tools are used by project managers to perform job submission tasks.

Job Submission Tool	Total (Number of Project Managers)
Jupyter Notebook	3
Custom (Script)	2
kubectl	2
oc	2
Ray	2
Custom (API)	1
Custom (ML model experimentation)	1
Custom (Job submission CLI)	1
GitHub/GitLab	1
k9s	1
Kubeflow Pipelines	1
kustomize	1
Python SDK	1
Spark	1

Appendix C

Codebook

In order to share our codebook as part of this dissertation, the 85 codes were organized into Tables C.1, C.2, C.3, C.4, and C.5.

Table C.1. 11 codes related to development processes included in our final codebook.

Codes Related to Development Processes
Agile process
Application versioning process
Dataset management process
Debugging process
Inventing process
Job submission process
Learning process
Note taking process
Onboarding process
Sharing process
Steady state process

Table C.2. 14 codes related to the frequency of performing a task (12) and using Kubernetes itself (1) as well as the time spent learning new concepts (1) included in our final codebook.

Codes Related to Frequency
Application switching frequency
Dataset management task frequency
Debugging task frequency
Image building task frequency
Job submission task frequency
Kubernetes frequency
ML/application source code task frequency
Monitoring task frequency
Observing the cluster task frequency
Onboarding task frequency
Task switching frequency
Time spent learning new concepts
Tool switching frequency
YAML/deployment logic task frequency

Table C.3. 20 codes related to tools included in our final codebook.

Codes Related to Tools
Application versioning tools
Cluster management tools
Dataset management tools
Debugging tools
Image building tools
Job submission tools
ML/application source code tools
Monitoring tools
Note taking tools
Observing the cluster tools
Onboarding tools
Pair programming tools
Team communication tools
Time spent building tools
Tool abstraction
Tool automation
Tool integration
Tool sharing
Tool/application source code language
YAML/deployment logic tools

Table C.4. 20 codes related to the gulfs of execution (10) and evaluation (10) included in our final codebook.

Codes Related to Gulfs of Execution and Evaluation
Gulf of evaluating application versioning
Gulf of evaluating dataset management
Gulf of evaluating debugging
Gulf of evaluating image building
Gulf of evaluating job submission
Gulf of evaluating ML/application source code
Gulf of evaluating monitoring
Gulf of evaluating observing the cluster
Gulf of evaluating onboarding
Gulf of evaluating YAML/deployment logic
Gulf of executing application versioning
Gulf of executing dataset management
Gulf of executing debugging
Gulf of executing image building
Gulf of executing job submission
Gulf of executing ML/application source code
Gulf of executing monitoring
Gulf of executing observing the cluster
Gulf of executing onboarding
Gulf of executing YAML/deployment logic

Table C.5. 20 other codes included in our final codebook.

Other Codes
Application versioning differences
Cost of operating platform
Dataset management differences
Development experience
Job roles
Job submission differences
Kubernetes comfortability
Kubernetes experience
Negative
Number of GPUs in the Kubernetes cluster
Number of Kubernetes applications
Number of Kubernetes nodes in the cluster
Number of Kubernetes pods in the cluster
Number of physical locations that provide Kubernetes nodes
Number of users onboarded
Onboarding differences
Positive
Prior to using Kubernetes
Team size
Technology changes

Bibliography

- [1] Introduction to infiniband. Nvidia - InfiniBand Technical Overview. URL https://network.nvidia.com/pdf/whitepapers/IB_Intro_WP_190.pdf. Accessed July 11, 2026.
- [2] Isoiec 25010: 2011 systems and software engineering-systems and software quality requirements and evaluation (square)-system and software quality models. Technical report, International Organization for Standardization., 2011.
- [3] Load balancer as a service (lbaas). OpenStack Documentation - Configuration, August 2019. URL <https://docs.openstack.org/mitaka/networking-guide/config-lbaas.html>. Accessed July 11, 2026.
- [4] Cncf survey 2020. Cloud Native Computing Foundation (CNCF) - Online Program, 2020. URL https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf. Accessed July 11, 2026.
- [5] Build, train, and deploy machine learning and deep learning models on floydhub. floydhub - floyd-docs, June 2020. URL <https://github.com/floydhub>. Accessed July 11, 2026.
- [6] Oasis topology and orchestration specification for cloud applications (tosca) tc. OASIS Open - TOSCA TC, 2020. URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca. Accessed July 11, 2026.
- [7] Convenience sampling method: How and when to use it? Qualtrics - Strategy and Research, November 2021. URL <https://www.qualtrics.com/experience-management/research/convenience-sampling>. Accessed July 11, 2026.
- [8] Cloud native 2023: The undisputed infrastructure of global technology. Cloud Native Computing Foundation (CNCF) - Reports, 2023. URL <https://www.cncf.io/reports/cncf-annual-survey-2023>. Accessed July 11, 2026.
- [9] Storage in hpc environment. Universiti Malaya - DICC, August 2023. URL <https://www.dicc.um.edu.my/2023/08/10/storage-in-hpc-environment/>. Accessed July 11, 2026.
- [10] Cloudify. Cloudify - About Cloudify, 2024. URL <https://docs.cloudify.co>. Accessed July

11, 2026.

- [11] Using cloud functions. IBM Cloud - Product guide, March 2024. URL <https://cloud.ibm.com/docs/cloud-object-storage?topic=cloud-object-storage-functions>. Accessed July 11, 2026.
- [12] Ansible vs. kubernetes: how they work together. Red Hat - Cloud automation, May 2025. URL <https://www.redhat.com/en/topics/automation/Ansible-vs-Kubernetes>. Accessed July 11, 2026.
- [13] Cloud native 2024: Approaching a decade of code, cloud, and change. Cloud Native Computing Foundation (CNCF) - Report/Survey, April 2025. URL <https://www.cncf.io/reports/cncf-annual-survey-2024>. Accessed July 11, 2026.
- [14] What is infrastructure as code (iac)? Red Hat - Automation and management, June 2025. URL <https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac>. Accessed July 11, 2026.
- [15] Search hacker news. Hacker News - Search, 2026. URL <https://hn.algolia.com>. Accessed July 11, 2026.
- [16] Amazon rekognition. AWS - Artificial Intelligence, 2026. URL <https://aws.amazon.com/rekognition>. Accessed July 11, 2026.
- [17] Ansible community documentation. Ansible documentation, 2026. URL <https://docs.ansible.com>. Accessed July 11, 2026.
- [18] Aws serverless application repository. AWS - Serverless, 2026. URL <https://aws.amazon.com/serverless/serverlessrepo>. Accessed July 11, 2026.
- [19] Scale with the largest, most extensive global infrastructure. AWS, 2026. URL <https://aws.amazon.com>. Accessed July 11, 2026.
- [20] Azure machine learning: Use an enterprise-grade ai service for the end-to-end machine learning (ml) lifecycle. Microsoft - Azure, 2026. URL <https://azure.microsoft.com/en-us/products/machine-learning>. Accessed July 11, 2026.
- [21] Azure documentation: Learn how to build and manage powerful applications using microsoft azure cloud services. Microsoft - Learn Azure, 2026. URL <https://learn.microsoft.com/en-us/azure/?product=popular>. Accessed July 11, 2026.
- [22] Azure vision in foundry tools. Microsoft - Azure, 2026. URL <https://azure.microsoft.com/en-au/products/ai-services/ai-vision>. Accessed July 11, 2026.

- [23] Azure functions. Microsoft - Functions, 2026. URL <https://azure.microsoft.com/en-us/products/functions>. Accessed July 11, 2026.
- [24] From data warehouse to autonomous data and ai platform. Google Cloud - BigQuery, 2026. URL <https://cloud.google.com/bigquery>. Accessed July 11, 2026.
- [25] Paas vs. iaas vs. saas vs. caas: How are they different? Google Cloud - Topics, 2026. URL <https://cloud.google.com/learn/paas-vs-iaas-vs-saas?hl=en>. Accessed July 11, 2026.
- [26] Cassandra. Apache - Cassandra, July 2026. URL <https://github.com/apache/cassandra>. Accessed July 11, 2026.
- [27] Powerful policy-based configuration management system software. Progress Chef - Chef Infra, 2026. URL <https://www.chef.io/products/chef-infra>. Accessed July 11, 2026.
- [28] Serverless computing vs. containers — how to choose. Cloudflare - Learning Center, 2026. URL <https://www.cloudflare.com/learning/serverless/serverless-vs-containers/>. Accessed July 11, 2026.
- [29] What is vendor lock-in? — vendor lock-in and cloud computing. Cloudflare - Learning Center, 2026. URL <https://www.cloudflare.com/learning/cloud/what-is-vendor-lock-in/>. Accessed July 11, 2026.
- [30] The proven development platform for cloud-native applications. Cloud Foundry, 2026. URL <https://www.cloudfoundry.org>. Accessed July 11, 2026.
- [31] Vision ai. Google Cloud - Cloud Vision API, 2026. URL <https://cloud.google.com/vision?hl=en>. Accessed July 11, 2026.
- [32] Cncf projects are the foundation of cloud and ai native computing. Cloud Native Computing Foundation (CNCF) - About, 2026. URL <https://www.cncf.io>. Accessed July 11, 2026.
- [33] The cncf annual cloud native survey: The infrastructure of ai’s future. Cloud Native Computing Foundation (CNCF) - Report, January 2026. URL <https://www.cncf.io/reports/the-cncf-annual-cloud-native-survey/>. Accessed July 11, 2026.
- [34] How can we help? Docker - Docs, 2026. URL <https://docs.docker.com>. Accessed July 11, 2026.
- [35] gcloud cli overview. Google Cloud SDK - Documentation, July 2026. URL <https://cloud.google.com/sdk/gcloud>. Accessed July 11, 2026.
- [36] Integrating github with slack. GitHub Docs - Integrations, 2026. URL <https://docs.github>.

- com/en/integrations/how-tos/slack/integrate-github-with-slack. Accessed July 10, 2026.
- [37] Glusterfs documentation. Gluster - Docs, 2026. URL <https://docs.gluster.org/en/latest>. Accessed July 11, 2026.
- [38] The new way to cloud. Google Cloud, 2026. URL <https://cloud.google.com/?hl=en>. Accessed July 11, 2026.
- [39] Cloud run functions. Google Cloud - Cloud Run functions, 2026. URL <https://cloud.google.com/functions>. Accessed July 11, 2026.
- [40] Google hangouts. Google - Chat, 2026. URL <https://hangouts.google.com>. Accessed July 11, 2026.
- [41] Google meet: Secure video conferencing for everyone. Google - Meet, 2026. URL <https://meet.google.com/landing>. Accessed July 11, 2026.
- [42] Hacker news. Y Combinator - News, 2026. URL <https://news.ycombinator.com>. Accessed July 11, 2026.
- [43] Hbase. Apache - HBase, July 2026. URL <https://github.com/apache/hbase>. Accessed July 11, 2026.
- [44] Hadoop. Apache - Hadoop, July 2026. URL <https://github.com/apache/hadoop>. Accessed July 11, 2026.
- [45] The cloud application platform for building, deploying, and scaling apps. Heroku, 2026. URL <https://www.heroku.com>. Accessed July 11, 2026.
- [46] Ibm watson visual recognition. IBM MediaCenter, 2026. URL https://mediacenter.ibm.com/media/IBM+Watson+Visual+Recognition/0_jbsmp6lq. Accessed July 11, 2026.
- [47] Jupyterlab documentation. JupyterLab - Read the Docs, 2026. URL <https://jupyterlab.readthedocs.io/en/latest>. Accessed July 11, 2026.
- [48] Kubeflow's superfood for data scientists. kubeflow - kale, 2026. URL <https://github.com/kubeflow-kale/kale>. Accessed July 11, 2026.
- [49] Keras 3 api documentation. Keras - API, 2026. URL <https://keras.io/api>. Accessed July 11, 2026.
- [50] Topic modeling. MALLET - Topics, 2026. URL <https://mimno.github.io/Mallet/topics>. Accessed July 11, 2026.

- [51] Mlflow documentation. mlflow - docs, 2026. URL <https://mlflow.org/docs/latest/>. Accessed July 11, 2026.
- [52] Basics of running jobs. NERSC Documentation - Running Jobs, July 2026. URL <https://docs.nersc.gov/jobs/>. Accessed July 11, 2026.
- [53] Nih hpc systems object store. NIH HPC - Object storage, 2026. URL <https://hpc.nih.gov/storage/object.html>. Accessed July 11, 2026.
- [54] About nngroup. NNG - About, 2026. URL <https://www.nngroup.com/about/>. Accessed July 11, 2026.
- [55] About nvivo: The premier software for qualitative data analysis. NVivo - About, 2026. URL <https://help-nv.qsrinternational.com/14/win/Content/about-nvivo/about-nvivo.htm>. Accessed July 11, 2026.
- [56] Build on the openai api platform. OpenAI - Platform, 2026. URL <https://platform.openai.com/docs/overview>. Accessed July 11, 2026.
- [57] What is openstack? OpenStack - Software, 2026. URL <https://www.openstack.org/software/>. Accessed July 11, 2026.
- [58] pandas documentation. pydata - pandas, June 2026. URL <https://pandas.pydata.org/docs/>. Accessed July 11, 2026.
- [59] Secure, scalable automation built for the modern enterprise. Perforce - Puppet, 2026. URL <https://www.puppet.com>. Accessed July 11, 2026.
- [60] Pytorch documentation. pytorch - docs, 2026. URL <https://docs.pytorch.org/docs/stable/index.html>. Accessed July 11, 2026.
- [61] Installing openshift container platform on bare metal. Red Hat Documentation - OpenShift Container Platform, 2026. URL https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/installing_on_bare_metal/index. Accessed July 11, 2026.
- [62] Amazon sagemaker: The next generation of amazon sagemaker is the center for all your data, analytics, and ai. AWS - Analytics, 2026. URL <https://aws.amazon.com/sagemaker>. Accessed July 11, 2026.
- [63] scikit-learn: Machine learning in python. scikit-learn - stable, June 2026. URL <https://scikit-learn.org/stable/>. Accessed July 11, 2026.
- [64] Stack exchange data explorer (sede). Stack Exchange - Stack Overflow, 2026. URL <https://data.stackexchange.com/stackoverflow/queries>. Accessed July 11, 2026.

- [65] Skip logic. Qualtrics Platform - Question Behavior, 2026. URL <https://www.qualtrics.com/support/survey-platform/survey-module/question-options/skip-logic>. Accessed July 11, 2026.
- [66] Spark. Apache - Spark, July 2026. URL <https://github.com/apache/spark>. Accessed July 11, 2026.
- [67] Swarm mode. Docker Engine - Swarm mode, 2026. URL <https://docs.docker.com/engine/swarm>. Accessed July 11, 2026.
- [68] An end-to-end platform for machine learning. TensorFlow - Install, 2026. URL <https://www.tensorflow.org>. Accessed July 11, 2026.
- [69] Terraform. HashiCorp Developer - Terraform, 2026. URL <https://developer.hashicorp.com/terraform>. Accessed July 11, 2026.
- [70] Your favorite forms. now with ai automation. Typeform - AI forms & automation, 2026. URL <https://www.typeform.com>. Accessed July 11, 2026.
- [71] Zahra Shakeri Hossein Abad, Oliver Karras, Kurt Schneider, Ken Barker, and Mike Bauer. Task interruption in software development projects: What makes some interruptions more disruptive than others? In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 122–132, 2018.
- [72] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210. IEEE, 2019.
- [73] Amritanshu Agrawal, Wei Fu, and Tim Menzies. What is wrong with topic modeling? and how to fix it using search-based software engineering. *Information and Software Technology*, 98:74–88, 2018.
- [74] Md Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. Classifying stack overflow posts on api issues. In *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*, pages 244–254. IEEE, 2018.
- [75] Syed Ahmed and Mehdi Bagherzadeh. What do concurrency developers ask about? a large-scale study using stack overflow. In *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, pages 1–10, 2018.
- [76] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*,

pages 2623–2631, 2019.

- [77] Moayad Alshangiti, Hitesh Sapkota, Pradeep K Murukannaiah, Xumin Liu, and Qi Yu. Why is developing machine learning applications challenging? a study on stack overflow posts. In *2019 acm/ieee international symposium on empirical software engineering and measurement (esem)*, pages 1–11. IEEE, 2019.
- [78] Anthropic. Start building with claude. Claude Platform Docs. URL <https://platform.claude.com/docs/en/home>. Accessed June 26, 2026.
- [79] David Armstrong, Ann Gosling, John Weinman, and Theresa Marteau. The place of interrater reliability in qualitative research: An empirical study. *Sociology*, 31(3):597–606, 1997.
- [80] Diana Arroyo, Paul Castro, Cora Coleman, Thuan Doan, Nick Mitchell, Sara Kokkila Schumacher, Ed Seabolt, Aleksander Slominski, Ansu Varghese, and Lionel Villard. The cloud, like building and running go binaries. In *IEEE International Conference on Distributed Computing Systems*, 2025.
- [81] The Kubeflow Authors. All of kubeflow documentation. Kubeflow - Docs, April 2020. URL <https://www.kubeflow.org/docs/>. Accessed July 11, 2026.
- [82] The Kubernetes Authors. Scheduling framework. Kubernetes Documentation - Scheduling, Preemption and Eviction, July 2025. URL <https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/>. Accessed July 11, 2026.
- [83] The Kubernetes Authors. Installing kubernetes with deployment tools. Kubernetes Documentation - Production environment, 2026. URL <https://kubernetes.io/docs/setup/production-environment/tools/>. Accessed July 11, 2026.
- [84] The Kubernetes Authors. Kubernetes documentation. Kubernetes Documentation - Documentation, 2026. URL <https://kubernetes.io/docs/home>. Accessed July 11, 2026.
- [85] The Kubernetes Authors. Event is a report of an event somewhere in the cluster. Kubernetes Documentation - Event, 2026. URL <https://kubernetes.io/docs/reference/kubernetes-api/cluster-resources/event-v1/>. Accessed July 11, 2026.
- [86] Mehdi Bagherzadeh and Raffi Khatchadourian. Going big: a large-scale study on what big data developers ask. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 432–442, 2019.
- [87] Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the 15th*

international conference on mining software repositories, pages 319–330, 2018.

- [88] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical software engineering*, 19(3):619–654, 2014.
- [89] Stefanie Beyer, Christian Macho, Martin Pinzger, and Massimiliano Di Penta. Automatically classifying posts into question categories on stack overflow. In *Proceedings of the 26th Conference on Program Comprehension*, pages 211–221, 2018.
- [90] John B Biggs and Kevin F Collis. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic press, 2014.
- [91] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [92] Benjamin S Bloom, Max D Engelhart, Edward J Furst, Walker H Hill, David R Krathwohl, et al. *Taxonomy of educational objectives: The classification of educational goals. Handbook 1: Cognitive domain*. Longman New York, 1956.
- [93] Lars Bratthall and Magne Jørgensen. Can you trust a single data source exploratory software engineering case study? *Empirical Software Engineering*, 7(1):9–26, 2002.
- [94] Gusseppe Bravo-Rocca, Peini Liu, Jordi Guitart, Ajay Dholakia, David Ellison, Jeffrey Falkanger, and Miroslav Hodak. Scanflow: A multi-graph framework for machine learning workflow management, supervision, and debugging. *Expert Systems with Applications*, 202:117232, 2022.
- [95] Ian Buchanan. How infrastructure as code (iac) manages complex infrastructures. Atlassian - Microservices Topics, 2026. URL <https://www.atlassian.com/microservices/cloud-computing/infrastructure-as-code>. Accessed July 11, 2026.
- [96] Raluca Budiu. Memory recognition and recall in user interfaces. NNG - Articles & Resources, January 2024. URL <https://www.nngroup.com/articles/recognition-and-recall/>. Accessed July 11, 2026.
- [97] Brendan Burns. The history of kubernetes & the community behind it, January 2026. URL <https://kubernetes.io/blog/2018/07/20/the-history-of-kubernetes-the-community-behind-it/>. Accessed June 26, 2026.
- [98] Brendan Burns, Eddie Villalba, Dave Streb, and Lachlan Evenson. *Kubernetes best practices: blueprints for building successful applications on Kubernetes*. ” O’Reilly Media, Inc.”, 2023.

- [99] Ben Caldwell, Michael Cooper, Loretta Guarino Reid, Gregg Vanderheiden, Wendy Chisholm, John Slatin, and Jason White. Web content accessibility guidelines (wcag) 2.0. *WWW Consortium (W3C)*, 290(1-34):5–12, 2008.
- [100] John L Campbell, Charles Quincy, Jordan Osserman, and Ove K Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological methods & research*, 42(3):294–320, 2013.
- [101] Kathy Charmaz. Constructing grounded theory (introducing qualitative methods series). *Constr. grounded theory*, 2014.
- [102] Zhenpeng Chen, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, Tao Xie, and Xuanzhe Liu. A comprehensive study on challenges in deploying deep learning based software. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 750–762, 2020.
- [103] Zhenpeng Chen, Huihan Yao, Yiling Lou, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, and Xuanzhe Liu. An empirical study on deployment faults of deep learning based mobile applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 674–685. IEEE, 2021.
- [104] Jürgen Cito, Philipp Leitner, Thomas Fritz, and Harald C Gall. The making of cloud applications: An empirical study on software development for the cloud. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 393–403, 2015.
- [105] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.
- [106] Stack Exchange Community. Stack exchange data dump, January 2014. URL <https://archive.org/details/stackexchange>. Accessed July 11, 2026.
- [107] Electron contributors. Electron: Build cross-platform desktop apps with javascript, html, and css. electronjs - Docs, 2026. URL <https://www.electronjs.org>. Accessed July 11, 2026.
- [108] Juliet Corbin and Anselm Strauss. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications, 2014.
- [109] Alex Cummaudo, Rajesh Vasa, Scott Barnett, John Grundy, and Mohamed Abdelrazek. Interpreting cloud computer vision pain-points: A mining study of stack overflow. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1584–1596, 2020.

- [110] Yasmin Uchôa da Silva, Gutemberg Borges França, Heloisa Musetti Ruivo, and Haroldo Fraga de Campos Velho. Forecast of convective events via hybrid model: Wrf and machine learning algorithms. *Applied Computing and Geosciences*, 16:100099, 2022.
- [111] Vedran Dakić, Mario Kovač, and Jurica Slovinac. Evolving high-performance computing data centers with kubernetes, performance analysis, and dynamic workload placement based on machine learning scheduling. *Electronics*, 13(13):2651, 2024.
- [112] Matthew C Davis, Sangheon Choi, Amy Wei, Sam Estep, Brad A Myers, and Joshua Sunshine. Testloop: A process model describing human-in-the-loop software test suite generation. *ACM Transactions on Software Engineering and Methodology*, 2025.
- [113] Daniele De Sensi, Tiziano De Matteis, Konstantin Taranov, Salvatore Di Girolamo, Tobias Rahn, and Torsten Hoefler. Noise in the clouds: Influence of network performance variability on application scalability. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–27, 2022.
- [114] MLOps: Continuous delivery and automation pipelines in machine learning. Kazmierczak, jarek and salama, khalid and huerta, valentin and kumar jang bahadur, sunil. Google Cloud Documentation - Cloud Architecture Center, August 2024. URL <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>. Accessed July 11, 2026.
- [115] Dask Developers. Dask python library for parallel and distributed computing. Dask - How to Use, 2026. URL <https://docs.dask.org/en/stable/>. Accessed July 11, 2026.
- [116] Jessica Díaz, Jorge Pérez, Carolina Gallardo, and Ángel González-Prieto. Applying inter-rater reliability and agreement in collaborative grounded theory studies in software engineering. *Journal of Systems and Software*, 195:111520, 2023.
- [117] Steve M Easterbrook and Timothy C Johns. Engineering the software for understanding climate change. *Computing in science & engineering*, 11(6):65–74, 2009.
- [118] Floris MA Erich, Chintan Amrit, and Maya Daneva. A qualitative study of devops usage in practice. *Journal of software: Evolution and Process*, 29(6):e1885, 2017.
- [119] Matt Farnia. Sig apps: build apps for and operate them in kubernetes. Kubernetes Blog, August 2016. URL <https://kubernetes.io/blog/2016/08/sig-apps-running-apps-in-kubernetes>. Accessed July 11, 2026.
- [120] Matt Farnia. Kubernetes application survey 2018. Kubernetes Blog, April 2018. URL <https://kubernetes.io/blog/2018/04/24/kubernetes-application-survey-results-2018/>. Accessed July 11, 2026.

- [121] Sally Fincher and Josh Tenenber. Making sense of card sorting data. *Expert Systems*, 22(3):89–93, 2005.
- [122] Kim Flaherty. Contextual inquiry: Inspire design by observing and interviewing users in their context. NNG - Articles & Resources, December 2020. URL <https://www.nngroup.com/articles/contextual-inquiry/>. Accessed July 11, 2026.
- [123] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. *Statistical methods for rates and proportions*. John Wiley & Sons, 2013.
- [124] John M Ford. Content analysis: An introduction to its methodology. *Personnel Psychology*, 57(4):1110, 2004.
- [125] Álvaro López García, Jesús Marco De Lucas, Marica Antonacci, Wolfgang Zu Castell, Mario David, Marcus Hardt, Lara Lloret Iglesias, Germán Moltó, Marcin Plociennik, Viet Tran, et al. A cloud-based framework for machine learning workloads and applications. *IEEE access*, 8:18681–18692, 2020.
- [126] D Randy Garrison, Martha Cleveland-Innes, Marguerite Koole, and James Kappelman. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The internet and higher education*, 9(1):1–8, 2006.
- [127] Johnu George, Ce Gao, Richard Liu, Hou Gang Liu, Yuan Tang, Ramdoot Pydipaty, and Amit Kumar Saha. A scalable and cloud-native hyperparameter tuning system. *arXiv preprint arXiv:2006.02085*, 2020.
- [128] Barney G Glaser, Anselm L Strauss, and Elizabeth Strutzel. The discovery of grounded theory; strategies for qualitative research. *Nursing research*, 17(4):364, 1968.
- [129] Dejan Golubovic and Ricardo Rocha. Training and serving ml workloads with kubeflow at cern. In *EPJ Web of Conferences*, volume 251, page 02067. EDP Sciences, 2021.
- [130] Ángel González-Prieto, Jorge Perez, Jessica Diaz, and Daniel López-Fernández. Reliability in software engineering qualitative research through inter-coder agreement. *Journal of Systems and Software*, 202:111707, 2023.
- [131] Eugene Gorelik. *Cloud computing models*. PhD thesis, Massachusetts Institute of Technology, 2013.
- [132] John Gruber. Markdown is a text-to-html conversion tool for web writers. Daring Fireball - Projects, December 2004. URL <https://daringfireball.net/projects/markdown>. Accessed July 11, 2026.
- [133] Michele Guerriero, Martin Garriga, Damian A Tamburri, and Fabio Palomba. Adoption,

- support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE International conference on software maintenance and evolution (ICSME)*, pages 580–589. IEEE, 2019.
- [134] Bruce Hanington and Bella Martin. Universal methods of design: 100 ways to research complex problems. *Develop innovative ideas, and design effective solutions*, 2012.
 - [135] Mubin Ul Haque, Leonardo Horn Iwaya, and M Ali Babar. Challenges in docker development: A large-scale study using stack overflow. In *Proceedings of the 14th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*, pages 1–11, 2020.
 - [136] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.
 - [137] Red Hat. Chapter 1. web console overview. Red Hat Documentation - OpenShift Container Platform. URL https://docs.redhat.com/en/documentation/openshift_container_platform/4.22/html/web_console/web-console-overview. Accessed July 29, 2026.
 - [138] Rashina Hoda, James Noble, and Stuart Marshall. Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Software Engineering*, 17(6):609–639, 2012.
 - [139] Randy Hodson. *Analyzing documentary accounts*. Number 128. Sage, 1999.
 - [140] Jim Holdsworth and Annie Badman. What is infrastructure as code (iac)? IBM - Infrastructure as Code, September 2025. URL <https://www.ibm.com/think/topics/infrastructure-as-code>. Accessed July 11, 2026.
 - [141] Karen Holtzblatt and Hugh Beyer. *Contextual design: evolved*. Morgan & Claypool Publishers, 2014.
 - [142] Edwin L Hutchins, James D Hollan, and Donald A Norman. Direct manipulation interfaces. *Human-Computer Interaction*, 1(4):311–338, 1985.
 - [143] What is containerization? Susnjara, stephanie and smalley, ian. IBM - Think, May 2024. URL <https://www.ibm.com/think/topics/containerization>. Accessed July 11, 2026.
 - [144] Sissie Hsiao James Manyika. An overview of the gemini app. Google Gemini - Overview, July 2024. URL <https://gemini.google/overview/>. Accessed June 26, 2026.
 - [145] Päivi Kinnunen and Beth Simon. Phenomenography and grounded theory as research methods in computing education research field. *Computer Science Education*, 22(2):

199–218, 2012.

- [146] Kubernetes Batch Working Group. <https://github.com/kubernetes/community/tree/master/wg-batch>.
- [147] Kueue: Kubernetes-native Job Queueing. <https://kueue.sigs.k8s.io/>.
- [148] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.
- [149] Chun-Hsiang Lee, Zhaofeng Li, Xu Lu, Tiyun Chen, Saisai Yang, and Chao Wu. Multi-tenant machine learning platform based on kubernetes. In *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*, pages 5–12, 2020.
- [150] Philipp Leitner, Erik Wittern, Josef Spillner, and Waldemar Hummer. A mixed-method empirical study of function-as-a-service software development in industrial practice. *Journal of Systems and Software*, 149:340–359, 2019.
- [151] Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao, Jidong Ge, and Zhihao Shan. A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, 157:110380, 2019.
- [152] Richard J Light. Measures of response agreement for qualitative data: some generalizations and alternatives. *Psychological bulletin*, 76(5):365, 1971.
- [153] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.
- [154] Eric S Liu, Dylan A Lukes, and William G Griswold. Refactoring in computational notebooks. *ACM Transactions on Software Engineering and Methodology*, 32(3):1–24, 2023.
- [155] Peini Liu and Jordi Guitart. Fine-grained scheduling for containerized hpc workloads in kubernetes clusters. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 275–284. IEEE, 2022.
- [156] Peini Liu, Gusseppe Bravo-Rocca, Jordi Guitart, Ajay Dholakia, David Ellison, and Miroslav Hodak. Scanflow-k8s: Agent-based framework for autonomic management and supervision of ml workflows in kubernetes clusters. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 376–385. IEEE, 2022.

- [157] Yan Liu, Zhijing Ling, Boyu Huo, Boqian Wang, Tianen Chen, and Esma Mouine. Building a platform for machine learning operations from open source frameworks. *IFAC-PapersOnLine*, 53(5):704–709, 2020.
- [158] Zeyang Liu, Jing Zhang, Yadong Yang, Yaping Wang, Wangjun Luo, and Xiancun Zhou. Enhancing weather forecast accuracy through the integration of wrf and bp neural networks: A novel approach. *Earth and Space Science*, 11(10):e2024EA003613, 2024.
- [159] Yiling Lou, Zhenpeng Chen, Yanbin Cao, Dan Hao, and Lu Zhang. Understanding build issue resolution in practice: symptoms and fix patterns. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 617–628, 2020.
- [160] Jakob Lüttgau, Michael Kuhn, Kira Duwe, Yevhen Alforov, Eugen Betke, Julian Kunkel, and Thomas Ludwig. Survey of storage systems for high-performance computing. *Supercomputing Frontiers and Innovations*, 5(1), 2018.
- [161] Christian Meske, Tobias Hermanns, Esther Von der Weiden, Kai-Uwe Loser, and Thorsten Berger. Vibe coding as a reconfiguration of intent mediation in software development: Definition, implications, and research agenda. *IEEE Access*, 13:213242–213259, 2025.
- [162] Microsoft. Visual studio 2026 release notes. Azure Visual Studio - Release Notes, June 2026. URL <https://learn.microsoft.com/en-us/visualstudio/releases/2026/release-notes>. Accessed June 26, 2026.
- [163] Antonio Molner and Alberto Guillen. Snapperml is a framework for machine learning. SnapperML - SnapperML, April 2026. URL https://github.com/SnapperML/SnapperML/blob/master/docs/source/experiment_specification.rst. Accessed July 11, 2026.
- [164] Antonio Molner, Francisco Carrillo-Perez, and Alberto Guillén. Snapperml: A python-based framework to improve machine learning operations. *SoftwareX*, 26:101648, 2024.
- [165] David Moreau, Kristina Wiebels, and Carl Boettiger. Containers for computational reproducibility. *Nature Reviews Methods Primers*, 3(1):50, 2023.
- [166] Robert G Newcombe. Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in medicine*, 17(8):873–890, 1998.
- [167] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158, 1994.
- [168] A Norman Donald. *The design of everyday things*. MIT Press, 2013.
- [169] OpenAI. Introducing chatgpt. OpenAI - Product, November 2022. URL <https://openai>.

com/index/chatgpt/. Accessed June 26, 2026.

- [170] Cliodhna O'Connor and Helene Joffe. Intercoder reliability in qualitative research: Debates and practical guidelines. *International journal of qualitative methods*, 19: 1609406919899220, 2020.
- [171] David B Palumbo. Programming language/problem-solving research: A review of relevant issues. *Review of educational research*, 60(1):65–89, 1990.
- [172] Sandipan Panda. Spotlight on sig apps. Kubernetes Blog, March 2025. URL <https://kubernetes.io/blog/2025/03/12/sig-apps-spotlight-2025/>. Accessed July 11, 2026.
- [173] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.
- [174] Witold Pedrycz. Why triangular membership functions? *Fuzzy sets and Systems*, 64(1): 21–30, 1994.
- [175] Dan Pilone and Russ Miles. *Head First Software Development*. O'Reilly Media, Inc., 2007.
- [176] Roel Popping. In search of one set of categories. *Quality and Quantity*, 26:147–155, 1992.
- [177] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. Deriving microservice code from underspecified domain models using devops-enabled modeling languages and model transformations. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 229–236. IEEE, 2020.
- [178] Biplob R Ray, Morshed Chowdhury, and Usman Atif. Is high performance computing (hpc) ready to handle big data? In *International Conference on Future Network Systems and Security*, pages 97–112. Springer, 2017.
- [179] Massimo Re Ferre and Chris Greenwood. Aws lambda for the containers developer. AWS Blogs - Containers, 2026. URL <https://aws.amazon.com/blogs/containers/aws-lambda-for-the-containers-developer/>. Accessed July 11, 2026.
- [180] Nathalie Riche, Anna Offenwanger, Frederic Gmeiner, David Brown, Hugo Romat, Michel Pahud, Nicolai Marquardt, Kori Inkpen, and Ken Hinckley. Ai-instruments: Embodying prompts as instruments to abstract & reflect graphical interface commands as general-purpose tools. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2025.
- [181] Kendra L Rieger. Discriminating among grounded theory approaches. *Nursing inquiry*,

26(1):e12261, 2019.

- [182] MS Riley, JG Greeno, JI Heller, and HP Ginsberg. Development of word problem solving ability. *Development of mathematical thinking*, pages 113–132, 1981.
- [183] Florent Robert, Hui-Yin Wu, Lucile Sassatelli, and Marco Winckler. The triangle of misunderstanding in interactive virtual narratives: Gulfs between system, designers and players. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2025.
- [184] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.
- [185] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. The programmer’s assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 491–514, 2023.
- [186] Johnny Saldaña. The coding manual for qualitative researchers. 2021.
- [187] Harini Sampath, Alice Merrick, and Andrew Macvean. Accessibility of command line interfaces. In *Proceedings of the 2021 CHI conference on human factors in computing systems*, pages 1–10, 2021.
- [188] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28, 2015.
- [189] Todd Sedano, Paul Ralph, and Cécile Péraire. Lessons learned from an extended participant observation grounded theory study. In *2017 IEEE/ACM 5th international workshop on conducting empirical studies in industry (CESI)*, pages 9–15. IEEE, 2017.
- [190] Himanshu Sharma, Manish Shrivastava, and Balwinder Singh. Physics informed deep neural network embedded in a chemical transport model for the amazon rainforest. *npj Climate and Atmospheric Science*, 6(1):28, 2023.
- [191] Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, 16(8):57–69, 1983.
- [192] Ben Shneiderman. A computer graphics system for polynomials. *Mathematics Teacher*, 67(2):111–113, 1974.

- [193] Ben Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology*, 1(3):237–256, 1982.
- [194] William Alexander Simm, Faiza Samreen, Richard Bassett, Maria Angela Ferrario, Gordon Blair, Jonathan Whittle, and PJ Young. Se in es: Opportunities for software engineering and cloud computing in environmental science. In *Proceedings of the 40th International conference on software engineering: software engineering in society*, pages 61–70, 2018.
- [195] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, M. Barker, K. G. Duda, X. Y. Huang, W. Wang, and J. G. Powers. A description of the advanced research wrf version 3. Technical report, National Center for Atmospheric Research, 2008.
- [196] Adam Smith. An inquiry into the nature and causes of the wealth of nations: Volume one. London: printed for W. Strahan; and T. Cadell, 1776., 1776.
- [197] Antero Taivalsaari, Tommi Mikkonen, Cesare Pautasso, and Kari Systä. Full stack is not what it used to be. In *International conference on web engineering*, pages 363–371. Springer, 2021.
- [198] The Ray Team. Welcome to ray: An open source framework to build and scale your ml and python applications easily. Ray - Docs, 2026. URL <https://docs.ray.io/en/latest/index.html>. Accessed July 11, 2026.
- [199] Jeff Teper. The next chapter: Moving from skype to microsoft teams. Microsoft - Blog home, February 2025. URL <https://www.microsoft.com/en-us/microsoft-365/blog/2025/02/28/the-next-chapter-moving-from-skype-to-microsoft-teams>. Accessed July 11, 2026.
- [200] The Henry Ford Archive of American Innovation. Henry ford: Assembly line. URL <https://www.thehenryford.org/collections-and-research/digital-collections/sets/7139>. Accessed June 26, 2026.
- [201] Yijun Tian, Waii Ng, Jialiang Cao, et al. Geek talents: Who are the top experts on github and stack overflow? *Computers, Materials & Continua*, 61(2), 2019.
- [202] TorchX. <https://pytorch.org/torchx/latest/>.
- [203] Priyan Vaithilingam and Philip J Guo. Bespoke: Interactively synthesizing custom guis from command-line applications by demonstration. In *Proceedings of the 32nd annual ACM symposium on user interface software and technology*, pages 563–576, 2019.
- [204] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. The sky is not the limit: multitasking across github projects. In *Proceedings of the 38th International Conference on Software Engineering*,

pages 994–1005, 2016.

- [205] Maike Vollstedt. To see the wood for the trees: The development of theory from empirical interview data using grounded theory. *Approaches to Qualitative Research in Mathematics Education: Examples of Methodology and Methods*, pages 23–48, 2015.
- [206] Artem Voronkov, Leonardo A Martucci, and Stefan Lindskog. System administrators prefer command line interfaces, don’t they? an exploratory study of firewall interfaces. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 259–271, 2019.
- [207] Jinfeng Wen, Zhenpeng Chen, Yi Liu, Yiling Lou, Yun Ma, Gang Huang, Xin Jin, and Xuanzhe Liu. An empirical study on challenges of application development in serverless computing. In *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 416–428, 2021.
- [208] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, 2016.
- [209] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. An empirical study of common challenges in developing deep learning applications. In *2019 IEEE 30th international symposium on software reliability engineering (ISSRE)*, pages 104–115. IEEE, 2019.
- [210] Yuanliang Zhang, Haochen He, Owolabi Legunsen, Shanshan Li, Wei Dong, and Tianyin Xu. An evolutionary study of configuration design and implementation in cloud systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 188–200. IEEE, 2021.
- [211] Xin Zhou, Huang Huang, He Zhang, Xin Huang, Dong Shao, and Chenxin Zhong. A cross-company ethnographic study on software teams for devops and microservices: organization, benefits, and issues. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 1–10, 2022.