

University of Brasília

Institute of Exact Sciences
Department of Computer Science

Practical Adoption: Lessons Learned Building ML-Enabled Systems

Guilherme Daniel Fernandes da Silva

Thesis presented in partial fulfillment of the requirements for the degree of the Master of Science in Informatics

Advisor
Prof.a Dr.a Carla Rocha

Brasília
2026



University of Brasília

Institute of Exact Sciences
Department of Computer Science

Practical Adoption: Lessons Learned Building ML-Enabled Systems

Guilherme Daniel Fernandes da Silva

Thesis presented in partial fulfillment of the requirements for
the degree the Master of Science in Informatics

Prof.a Dr.a Carla Rocha (Advisor)
FCTE/UnB

Prof. Dr. Edna Dias Canedo
Universidade de Brasília

Prof. Dr. Gustavo Henrique Lima Pinto
Universidade Federal do Pará

Prof.a Dr.a Cláudia Nalon
Coordinator of the Graduate Program in Informatics

Brasília, May 27, 2026

Abstract

Context: The growing adoption of Machine Learning (ML) in software products has shifted development from explicit, rule-based logic toward data-driven systems whose behavior is learned and continues to change after deployment. This shift raises socio-technical challenges of quality assurance, governance, coordination, and requirements. Organizations adopting ML for the first time under limited resources often struggle to move promising prototypes into dependable production systems.

Objectives: This thesis investigates the challenges these organizations face when operationalizing ML for the first time (RQ1) and the minimal set of practices that enables them to deploy ML components to production (RQ2).

Methods: We conducted an exploratory qualitative multi-case study of two first-time adopters, Brasil Participativo, a Brazilian federal-government participation platform, and AdsRay, a disinformation-detection tool built by an academic laboratory together with a civil-society organization. We combined 9 semi-structured interviews, 67 technical artifacts, 16 meeting transcriptions, and participant observation. Using hybrid thematic analysis, we coded 302 quotations into eight themes, grouped the challenges into four dimensions, filed the practices under the challenge each addresses, and compared the observed practices against an established MLOps practice catalog.

Results: The challenges span data dependencies and contracts, resource and regulatory constraints, cross-team coordination, requirements uncertainty, domain knowledge, operationalization friction, monitoring and evaluation gaps, and delivery disruption. In response, the teams adopted practices such as inter-team data contracts, cross-functional coordination routines, local open-source models, use-case scoping, and human-in-the-loop validation. Some of these practices match documented MLOps practices, and some were not documented in the practice catalog we used. Both cases reached success at an entry level of operationalization, where a component leaves the experimental environment and runs as a served, data-fed, and maintained part of the product.

Conclusions: The study offers empirical evidence from real resource-constrained settings, a socio-technical framing of ML operationalization, an account of recurrent barriers, and a cross-case synthesis of practices, team structures, and artifacts that can help

first-time adopters bring ML to production and sustain it beyond a single project.

Keywords: MLOps; machine learning systems; software engineering; qualitative multi-case study; resource-constrained organizations; first-time ML adoption

Resumo estendido

Contexto. A crescente adoção de Aprendizado de Máquina (ML) em produtos de software mudou o paradigma de desenvolvimento de uma lógica baseada em regras explícitas, escritas por desenvolvedores, para sistemas orientados a dados, cujo comportamento é aprendido e continua a mudar após a implantação. Essa transição introduz desafios sociotécnicos de garantia de qualidade, governança, rastreabilidade, monitoramento e especificação de requisitos, além de exigir maior coordenação entre cientistas de dados, engenheiros de software, engenheiros de dados e demais partes interessadas. A literatura já documenta fontes recorrentes de dificuldade, como dívida técnica oculta, dependência de dados, deriva de conceito, baixa reprodutibilidade e o descompasso entre a avaliação *offline* e o comportamento em uso real. Abordagens de MLOps propõem articular práticas de engenharia de dados, engenharia de modelos, entrega e monitoramento contínuos. Alguns trabalhos descrevem os desafios e práticas de MLOps que podem ser adotadas para operacionalizar sistemas de ML [1, 2, 3, 4, 5]. Todavia, esses trabalhos apresentam os desafios e práticas na perspectiva de grandes organizações, não ficando claro quais práticas efetivamente sustentam a passagem da experimentação para a produção em organizações que adotam ML pela primeira vez.

Problema e objetivo. Organizações que adotam ML pela primeira vez frequentemente operam com infraestrutura modesta, equipes multidisciplinares pequenas e capacidade financeira limitada para experimentação e manutenção. Nesses contextos, um número reduzido de profissionais costuma acumular engenharia de dados, desenvolvimento de modelos, implantação e operação. Grande parte das lições disponíveis provém de grandes empresas e ambientes de engenharia maduros e pode não se transferir para organizações menores e iniciantes. Esta dissertação investiga esse problema por meio de duas questões de pesquisa: (RQ1) quais são os principais desafios enfrentados por organizações ao implementar e implantar aplicações de ML pela primeira vez em contextos com recursos limitados, e (RQ2) qual o conjunto mínimo de práticas de MLOps que viabiliza a implantação de sistemas de ML em produção nessas organizações.

Metodologia. Conduzimos um estudo de caso múltiplo, qualitativo e exploratório, seguindo as diretrizes de Runeson e Höst, sobre dois adotantes iniciantes. O primeiro

é a Brasil Participativo, plataforma de participação social do governo federal brasileiro, desenvolvida pela Universidade de Brasília em parceria com a Presidência da República e construída sobre o arcabouço de código aberto Decidim. O segundo é o AdsRay, ferramenta de detecção de desinformação em portais de notícias, desenvolvida pelo laboratório acadêmico Lab Livre em parceria com a organização da sociedade civil Sleeping Giants Brasil. A unidade de análise é a operacionalização de componentes de ML, e o desenho é incorporado (*embedded*), pois a Brasil Participativo contribui com três componentes (moderação de conteúdo, classificação e recomendação de propostas e geração de relatórios) e o AdsRay com um (o *pipeline* de classificação de desinformação). Combinamos nove entrevistas semiestruturadas, conduzidas remotamente e com duração média de 45 minutos, sessenta e sete artefatos técnicos, dezesseis transcrições de reuniões e observação participante. Na Brasil Participativo, entrevistamos um especialista de domínio, três engenheiros de ML e dois líderes técnicos de engenharia de dados. No AdsRay, um especialista de domínio e dois engenheiros de ML. A análise seguiu o processo de análise temática de Braun e Clarke, com codificação híbrida: os desafios foram codificados de forma indutiva, e as práticas partiram, de forma dedutiva, de um catálogo consolidado de práticas de MLOps, admitindo o surgimento de práticas ainda não documentadas. No total, foram codificadas 302 citações em 76 códigos, consolidados em oito temas. Os temas de desafio foram agrupados nas quatro dimensões de Amrit e Narayanappa (organizacional, técnica, operacional e de negócio), e as práticas adotadas foram associadas ao desafio que cada uma endereça. A triangulação entre entrevistas, documentação e observação buscou comparar o que os participantes relatam com o que está registrado nos artefatos e mitigar vieses.

Resultados. Os desafios identificados, organizados em oito temas, abrangem dependências e contratos de dados (T1), restrições de recursos e regulatórias (T2), coordenação entre equipes (T3), incerteza de requisitos (T4), conhecimento de domínio (T5), atritos na operacionalização de ML (T6), lacunas de monitoramento e avaliação (T7) e interrupções na entrega (T8). A dependência de pessoas-chave e as restrições de governança, segurança e soberania de dados, associadas ao contexto público e acadêmico, aparecem de forma marcante. A comparação entre os casos evidenciou padrões distintos. Por exemplo, o monitoramento e a avaliação (T7) foram o tema menos evidente na Brasil Participativo e o mais denso no AdsRay. Em resposta a esses desafios, as equipes adotaram práticas como contratos de dados entre equipes apoiados em tabelas Gold, coordenação multifuncional, uso de modelos locais de código aberto para preservar dados e reduzir custos, delimitação de casos de uso, calibração de critérios por meio de testes práticos, validação com humano no circuito em lote e *harnesses* de avaliação para sistemas de RAG. Parte dessas práticas corresponde a práticas já catalogadas na literatura de MLOps, e parte

não estava documentada no catálogo de práticas que utilizamos. Definimos o sucesso da operacionalização em um nível inicial, condizente com a maturidade dos casos. Um componente é considerado operacionalizado quando deixa o ambiente experimental e passa a ser alimentado por uma fonte de dados repetível, torna-se acessível por meio de uma API ou serviço implantado em vez de ser executado na máquina de um pesquisador, é executado em infraestrutura compartilhada, tem seu artefato versionado e recuperável, é integrado ao produto e pode ser operado por mais de uma pessoa.

Contribuições e considerações finais. A pesquisa oferece quatro contribuições. A primeira é um enquadramento conceitual que situa a operacionalização de ML em uma perspectiva sociotécnica. A segunda são evidências empíricas, obtidas em contextos reais com recursos limitados, sobre as práticas e os artefatos associados a um caminho bem-sucedido da experimentação à produção. A terceira é a identificação de barreiras recorrentes, incluindo atritos técnicos, restrições relacionadas a dados e problemas de coordenação entre papéis. A quarta é uma síntese, entre casos, de práticas, estruturas de equipe e artefatos capazes de apoiar equipes na implantação mais confiável de sistemas habilitados por ML. Em conjunto, os resultados indicam que, para organizações iniciantes e com recursos limitados, levar ML à produção apoia-se sobretudo em contratos de dados claros e coordenação entre equipes.

Palavras-chave: MLOps; sistemas de aprendizado de máquina; engenharia de software; estudo qualitativo de múltiplos casos; organizações com recursos limitados; adoção inicial de ML

Contents

1 Introduction **1**

1.1 Problem Outline 1
1.2 Research Questions 3
1.3 Claimed Contributions 3
1.4 Data Availability 4
1.5 Thesis Structure 4

2 Background **5**

2.1 From Software to ML Systems 5
2.2 DevOps 7
2.3 MLOps: DevOps applied in ML systems 9
2.3.1 Principles 11
2.3.2 Technical components 12
2.3.3 Roles 12
2.3.4 Architecture and workflow 13
2.3.5 Maturity and open challenges. 14
2.4 Challenges of Implementing MLOps 14
2.4.1 Organizational challenges 15
2.4.2 ML-system challenges 16
2.4.3 Operational challenges 16
2.4.4 Software Requirements 16
2.5 Related Work 19

3 Methodology **21**

3.1 Case Selection 21
3.2 Case A: Brasil Participativo 21
3.3 Case B: AdsRay 25
3.4 Positioning the cases on the MLOps maturity model 27
3.5 Study Design 28

3.5.1 Unit of Analysis . . . 29
3.5.2 Success Indicators . . . 30
3.6 Data Collection . . . 30
3.6.1 Document Collection . . . 31
3.6.2 Interviews . . . 32
3.6.3 Participant Observation . . . 36
3.6.4 Ethics and Consent . . . 36
3.7 Data Analysis . . . 36

4 Results **39**
4.1 Organizational challenges . . . 43
4.1.1 T3: Cross-Team Coordination Breakdowns . . . 43
4.1.2 T5: Domain Knowledge as an Adoption Barrier . . . 45
4.1.3 T8: Organizational Priorities Changing (Delivery Disruption) . . . 46
4.2 Technical challenges . . . 46
4.2.1 T1: Data Dependency and Contract Failures . . . 47
4.2.2 T6: ML Operationalization Friction . . . 48
4.3 Operational challenges . . . 49
4.3.1 T7: Monitoring and Evaluation Gaps . . . 49
4.4 Business challenges . . . 51
4.4.1 T2: Resource and Regulatory Constraints . . . 52
4.4.2 T4: Requirements Uncertainty and Late Validation . . . 53
4.5 Practices adopted across the challenges . . . 54

5 Discussion **57**
5.1 Organizational challenges . . . 57
5.1.1 Coordination as a team-structure effect (T3) . . . 57
5.1.2 Skill profile and the operationalization gap . . . 59
5.1.3 Domain knowledge as an adoption barrier (T5) . . . 60
5.1.4 Delivery disruption as a structural symptom (T8) . . . 60
5.2 Technical challenges . . . 61
5.2.1 Data dependency and contract failures (T1) . . . 61
5.2.2 Deployment and release friction (T6) . . . 62
5.3 Operational challenges . . . 62
5.3.1 Monitoring, evaluation, and the evaluation inversion (T7) . . . 62
5.4 Business challenges . . . 64
5.4.1 Resource and regulatory constraints (T2) . . . 64
5.4.2 Requirements uncertainty and late validation (T4) . . . 64

5.5 Practices adopted as the study's RQ2 contribution 64
5.6 Cross-case synthesis 65
5.7 Positioning against prior work and contribution 66
5.8 Answers to the research questions 67
5.9 Implications for research 67
5.10 Implications for practice 68
5.11 Implications for education 68
5.12 Recommendations and lessons learned 69

6 Conclusion **71**
6.1 Threats to Validity, Reliability, and Limitations 72
6.2 Future Work 73

References **74**

Appendix **80**

A Codebook **81**

B Meetings held by the Machine Learning team **86**

List of Figures

2.1 The nine-stage machine learning workflow 6
2.2 The DevOps lifecycle 8
2.3 Hidden technical debt in ML systems 10
2.4 MLOps at the intersection of ML, SE, and data engineering 11
2.5 Simplified end-to-end MLOps architecture 13
2.6 Taxonomy of MLOps implementation challenges 15
2.7 Requirements concerns mapped to MLOps principles and components . . . 17

3.1 Team structure of Case A 23
3.2 Case A ML products and their cross-team dependencies 23
3.3 Case A tooling 24
3.4 Team structure of Case B 25
3.5 Case B disinformation-detection pipeline 27
3.6 Case B tooling 27
3.7 Research design 29
3.8 The LLM-assisted analysis process 38
3.9 Typology of first-time MLOps adoption challenges 38

4.1 Thematic coding structure 41
4.2 Grounding per theme, by case 42
4.3 Challenge themes and practices adopted 55

5.1 Challenge–practice code co-occurrence 59

List of Tables

3.1 The two cases on the MLOps maturity model 28
3.2 Indicators of an operationalized ML component. 30
3.3 Document sources and selection rationale 31
3.4 Composition of the analyzed corpus 32
3.5 Interview protocol used in data collection 34
3.6 Inclusion criteria for participant selection. 35
3.7 Exclusion criteria for participant selection. 35

4.1 Anonymized participants and roles 40
4.2 Code frequency by dimension and theme 42
4.3 Practices added to the codebook 43

5.1 Recommendations for constrained, first-time MLOps adoption 69

A.1 Challenge and activity codes (RQ1) 81
A.2 First-order practice codes and their Alves et al. cross-reference 83

B.1 Meetings held on Machine Learning 86

Chapter 1

Introduction

The increasing adoption of Artificial Intelligence (AI), and of Machine Learning (ML) in particular, has expanded the scope of contemporary Software Engineering. While traditional software behaves according to rules developers write explicitly, ML-enabled systems learn important parts of their behavior from data and can keep changing after deployment. This changes the technical architecture of software products and the way teams design, validate, deploy, and maintain them over time [6, 7, 8].

These characteristics make ML-enabled systems substantially different from traditional software. Their probabilistic behavior, dependence on data quality, model opacity, and susceptibility to post-deployment change make development and validation harder than in traditional settings [9, 10, 8]. Adopting ML also introduces new roles, artifacts, and dependencies, and it requires closer coordination among data scientists, software engineers, data engineers, and product stakeholders [8]. Prior work has proposed processes and workflows for AI-based products, yet there is still no consolidated understanding of which practices most effectively support the move from experimentation to sustained production use, especially for organizations adopting ML for the first time [8, 11, 12].

1.1 Problem Outline

Despite the rapid adoption of ML in software products, many organizations still struggle to turn promising prototypes into dependable production systems. Success depends on more than model accuracy. Teams have to coordinate data preparation, experimentation, deployment, monitoring, retraining, and governance across several roles and artifacts [8, 13, 1]. These difficulties are sharper in organizations that adopt ML for the first time under tight resource constraints.

Such organizations usually operate with modest technical infrastructure, small multidisciplinary teams, and little financial room for experimentation and maintenance [3].

This constraint shapes the work in concrete ways. The teams are small and rarely include a dedicated MLOps specialist, so a handful of practitioners cover data engineering, model development, deployment, and operations at once, often without prior production experience [12, 14]. In these organizations, compute and infrastructure budgets are limited, which rules out the largest hosted models and pushes teams toward lighter and cheaper alternatives. On top of this, they face data governance, versioning, monitoring, and continuous-delivery demands that established MLOps processes are meant to handle but that these teams have not yet put in place [1, 11].

The literature already names several sources of difficulty, among them data dependence, hidden technical debt, drift, limited reproducibility, and the gap between offline evaluation and real-world behavior [13, 8, 15]. Much of this guidance, however, is drawn from large, well-resourced organizations with mature engineering environments and dedicated MLOps teams [1, 16], and it may not carry over to smaller organizations adopting ML for the first time [4]. What remains thin is empirical evidence on which concrete practices help such teams operationalize ML consistently, and on how data practices, engineering, MLOps capabilities, and organizational coordination interact in real projects.

To address this gap, this work presents a qualitative multi-case study of two organizations adopting machine learning *for the first time* under tight resource constraints. Brasil Participativo, developed from 2024 to 2026 by the University of Brasília with the Presidency of the Republic of Brazil, is a digital participation platform. AdsRay, built from 2025 to 2026 by the research laboratory Lab Livre together with the social organization Sleeping Giants Brasil, detects disinformation in news published on websites and in newspapers. In both, ML was new in every sense: the teams, the infrastructure, and the intended users were all encountering it for the first time, and neither project began with a dedicated MLOps specialist. Both operate with modest infrastructure, small multidisciplinary teams, limited operational capacity, and no established MLOps processes. Their public-sector and academic settings add further constraints on infrastructure provisioning, security policies, and data access, alongside strong concerns about data security and sovereignty and requirements that change often.

We examine how these organizations operationalize ML through a socio-technical lens that weighs technology, teams, and coordination together. By following each ML component from early experimentation through to production, we identify the challenges that arise during operationalization and the practices adopted to sustain a reliable path to it, and from this analysis we derive a set of practices, team structures, and artifacts that can help resource-constrained organizations adopt ML and build systems that endure beyond a single project.

1.2 Research Questions

To address this problem, the study is guided by the following research questions:

RQ1: What are the key challenges organizations face when implementing and deploying ML applications for the first time in resource-constrained settings?

This question examines the technical, organizational, and coordination issues that inhibit the transition from ML experimentation to production in organizations with limited infrastructure, small multidisciplinary teams, and no prior MLOps experience.

RQ2: Which minimal set of MLOps practices enables resource-constrained organizations to deploy ML systems to production?

Because such teams cannot adopt the full MLOps catalogue, this question investigates the smallest set of practices, artifacts, and collaboration mechanisms that carries an ML component from experimentation to production, prioritizing those that most reduce the risk of a failed deployment over comprehensive maturity.

1.3 Claimed Contributions

This study contributes at both the conceptual and practical levels. Grounded in 302 coded quotations from two first-time, resource-constrained adopters, the claimed contributions are:

- **C1 – Conceptual framing:** a characterization of first-time ML operationalization that organizes its challenges into the four dimensions of Amrit and Narayanappa [16] (organizational, technical, operational, and business), locates a resource-constrained ML organization on the topology ladder and identifies the horizontal *Platform team* as the structural remedy for its coordination deficit.
- **C2 – Empirical evidence on practices:** evidence, grounded in real organizational settings, of the practices that carry ML from experimentation to production under constraint, distilled into a repertoire of **19 practices** (the NP family) that extend the established MLOps practice codebook of Alves et al. [2].
- **C3 – Typology of barriers:** a typology of **eight recurrent challenge themes** spanning the four dimensions, from data-dependency and regulatory-access barriers to cross-team coordination and delivery disruption.
- **C4 – Cross-case synthesis:** an **challenge-to-practice mapping** linking each challenge theme to the practices adopted in response, and a cross-case synthesis of the two contrasting cases that surfaces where they diverge.

1.4 Data Availability

The data that support the findings of this study are available on Zonedo, and can be accessed through the following URL: .

1.5 Thesis Structure

This thesis is organized as follows. chapter 2 provides the theoretical background, covering the transition from traditional software to ML systems, the MLOps lifecycle and maturity perspective, and the main challenges reported in the literature for productionizing ML.

chapter 3 describes the methodology, justifies the adoption of a qualitative multi-case study, and details the case selection strategy, data collection procedures, and data analysis approach. The chapter also explains how document analysis, semi-structured interviews, and participant observation are combined to investigate the phenomenon.

chapter 4 reports the challenges and the practices adopted across the two cases, organized into themes. chapter 5 interprets these findings, positions them against prior work, and answers the research questions. chapter 6 summarizes the contributions, discusses limitations, and outlines directions for future work.

Chapter 2

Background

This section introduces the conceptual foundations of this work. We first contextualize the shift from Traditional Software to ML Systems, and then present DevOps, the software engineering paradigm from which Machine Learning Operations (MLOps) inherits most of its principles and practices. We then discuss why machine learning (ML) systems challenge established DevOps practices, and finally characterize MLOps in terms of its definition, principles, technical components, roles, and reference architecture.

2.1 From Software to ML Systems

Integrating Machine Learning into software applications represents a shift from deductive to inductive reasoning. Traditionally, Software Engineering (SE) followed a deductive development paradigm, in which developers specified system behavior using rule-based logic [8, 13]. Under this paradigm, requirements are translated into deterministic software structures (if-then statements and loops), enabling system behavior to be analyzed and validated through static analysis and test-based techniques [13].

In contrast, SE has evolved in recent years to meet new industry demands. Machine Learning (ML) systems have become increasingly common and have contributed to a shift toward an inductive development paradigm, in which behavior is learned from data [17, 8, 13]. Consequently, the primary artifacts that determine system behavior extend beyond source code and include training data, feature representations, and learned model parameters [8, 13].

This shift has two direct implications for SE work. First, many design choices become data-centered: decisions about data collection, labeling guidelines, and data curation may influence system behavior as much as implementation decisions [8]. Second, development becomes more socio-technical: data scientists, software engineers, and product stakeholders

must coordinate around shared artifacts (datasets, experiments, evaluation reports) to ensure reproducibility and maintainability across iterations.

Moreover, the model's decision logic is not directly represented in source code and cannot be inspected in the same way as traditional programs [8, 9]. In ML systems, even applications with correct code can still fail because of data drift and concept drift [8, 13, 15]. As a result, quality assurance shifts from purely deterministic test suites to a combination of offline evaluation, careful dataset management, and continuous validation after deployment.

Sculley et al. [13] argue that ML code is only a small fraction of the overall system. Much of the system consists of infrastructure for data handling, pipelines, integration, and operational support. ML systems require complex techniques, including configuring data pipelines, integrating various components, managing data dependencies, monitoring, and deploying systems in production.

ML system lifecycles differ substantially due to their data-centric nature. As a result, teams must adapt their processes for developing and delivering ML systems, changing how they manage code, data, and models [8, 18, 9, 2]. Amershi et al. [8] present a nine-stage development pipeline in Figure 2.1.

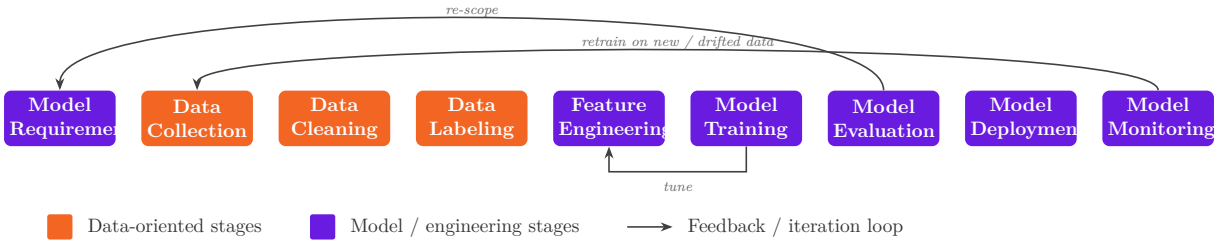


Figure 2.1: The nine-stage machine learning workflow [8].

Unlike the linear sequence presented in Figure 2.1, the process is highly iterative and relies on several feedback loops, which we represent with the feedback arrows in the figure. In this context, engineers may return to earlier stages to adjust inputs or parameters. We describe the ML workflow below:

- **Model Requirements:** software designers decide which features can be developed using ML and whether they apply to a new product or an existing one.
- **Data Collection:** teams evaluate and integrate available datasets or collect their own.
- **Data Cleaning:** engineers remove inaccurate or noisy records from the dataset.
- **Data Labeling:** data scientists assign ground-truth labels to each record.

- **Feature Engineering:** extraction and selection of informative features for ML models.
- **Model Training:** data scientists train models using the selected features.
- **Model Evaluation:** engineers evaluate the trained model on test or holdout datasets using predefined metrics.
- **Model Deployment:** engineers integrate the validated model into the production environment so that it can serve predictions.
- **Model Monitoring:** the team continuously tracks the deployed model, watching for errors and for data or concept drift that degrade its quality.

ML workflows rely primarily on experimentation to produce a suitable model for a given problem, unlike agile software processes, which rely primarily on feedback loops. In contexts where systems contain multiple integrated ML components that interact in complex and unpredictable ways, this workflow becomes even more complex. Various authors therefore argue that software engineering practices must be adapted to accommodate these differences [13, 19, 20]. These differences create operational risks and motivate the adoption of MLOps as a standard approach for managing the machine learning lifecycle.

2.2 DevOps

Over the past decades, several software process models and development methodologies have been proposed with the common goal of delivering production-ready software, from the classic waterfall model to agile methods. Although these processes and methodologies solved common delivery problems, they did not solve the conflict between development teams, pressured to deliver new features quickly, and operations teams, responsible for keeping systems stable and reliable [1, 21]. To address this problem, the concept of *DevOps* emerged around 2008/2009. The term originated at the 2008 Agile Conference in Toronto, where Patrick Debois highlighted the need to resolve this conflict when both teams had to collaborate to deliver quick responses to customer demands [21]. In a Delphi study with 42 IT experts, Iden et al. [22] systematically characterized the lack of cooperation between developers and IT operations personnel, including poor communication, insufficient knowledge transfer, and operations involvement too late in projects.

DevOps is a paradigm that addresses both social and technical issues in software-producing organizations [1]. In a systematic mapping study, Jabbari et al. [23] define DevOps as a development methodology that bridges the gap between development and

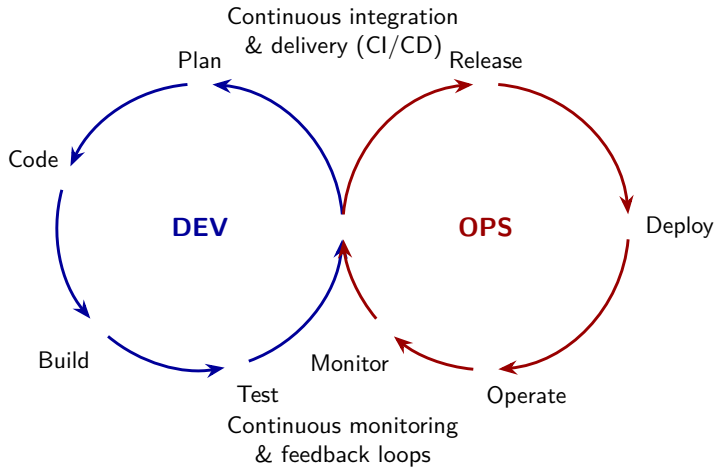


Figure 2.2: The DevOps lifecycle as a continuous Dev–Ops loop (own representation, based on [1, 25, 26]).

operations, emphasizing communication and collaboration, continuous integration, quality assurance, and automated deployment. Similarly, Díaz et al. [21] define DevOps as "a collaborative and multidisciplinary organizational effort to automate continuous delivery of new software updates while guaranteeing their correctness and reliability." Two complementary dimensions are recurrent across definitions: a *cultural* dimension, concerned with eliminating silos between development and operations through collaboration, communication, shared responsibility, and knowledge sharing, and a *technical* dimension, concerned with automating the path from a code change to production [21, 24]. Luz et al. [24] found, in a grounded theory study of DevOps adoption in real-world organizations, that a collaborative culture is the central enabler from which other DevOps capabilities, such as automation, sharing, and continuous measurement, derive.

On the technical side, DevOps applies automation through the practices of *continuous integration, continuous delivery, and continuous deployment* (CI/CD), enabling fast, frequent, and reliable releases, complemented by continuous testing, quality assurance, continuous monitoring, logging, and feedback loops [1]. These practices belong to the broader movement of *continuous software engineering* [26]. Following Shahin et al. [27], continuous integration means that developers merge their work frequently, with every change automatically built and tested. Continuous delivery keeps the software reliably releasable at any time after each change. Continuous deployment goes one step further by automatically deploying every change that passes the pipeline to production. Figure 2.2 depicts the resulting lifecycle as a continuous loop in which planning, coding, building, and testing activities (Dev) are seamlessly connected to releasing, deploying, operating, and monitoring activities (Ops), with monitoring insights flowing back into planning. This automation is supported by a rich tool ecosystem, which can be differentiated

into six groups [21, 28]: collaboration and knowledge sharing (e.g., Slack, GitLab wiki), source code management (e.g., GitHub, GitLab), build process (e.g., Maven), continuous integration (e.g., Jenkins, GitLab CI), deployment automation (e.g., Docker, Kubernetes), and monitoring and logging (e.g., Prometheus, Logstash).

DevOps adoption also has an organizational dimension. Companies reorganize their team structures to address silos, shared ownership, and cultural values, and several empirical taxonomies of such structures have been proposed. This reorganization rests on a long-standing observation. Conway's Law holds that a system's architecture tends to mirror the communication structure of the organization that builds it [29]. Misaligned or siloed teams therefore tend to produce fragmented, integration-heavy systems. Shahin et al. [30] identified four team configurations for continuous delivery and deployment, ranging from separate development and operations teams with closer collaboration to setups with no visible operations team. Leite et al. [25] distinguish siloed departments, classical DevOps, cross-functional teams, and platform teams. In contrast, López-Fernández et al. [31] characterize four analogous structures through variables such as collaboration frequency, ownership sharing, and autonomy. Díaz et al. [21] harmonized these partially overlapping taxonomies into a unified theory of DevOps team structures, showing that the different vocabularies in the literature describe largely consistent organizational patterns. Importantly, this body of work provides empirical evidence that breaking down silos improves software delivery performance, e.g., deployment frequency, lead time from commit to production, and mean time to recovery [25, 31], and that practicing DevOps improves software quality [32]. As a result, practitioners and academics have accumulated a wealth of experience in automating and operating software systems with DevOps, and this experience is now being transferred to the automation and operationalization of ML [1].

2.3 MLOps: DevOps applied in ML systems

Although ML-based systems are, in the end, software systems, applying DevOps practices to them is not straightforward, because their behavior is determined not only by source code but also by the data used to train the models. Sculley et al. [13] famously argued that ML systems incur massive *hidden technical debt*. According to them, only a small fraction of a real-world ML system is ML code, while the surrounding infrastructure (data collection, feature extraction, configuration, serving, and monitoring) is vast and prone to system-level anti-patterns (Figure 2.3). ML introduces debt sources without a direct analog in traditional software, such as entanglement ("changing anything changes everything"), unstable data dependencies, and feedback loops in which a deployed model influences its

own future training data [13]. Consequently, established testing and monitoring practices must be extended with data validation, model validation, and production monitoring of predictive quality [33], and the data lifecycle itself becomes a first-class engineering concern [34].

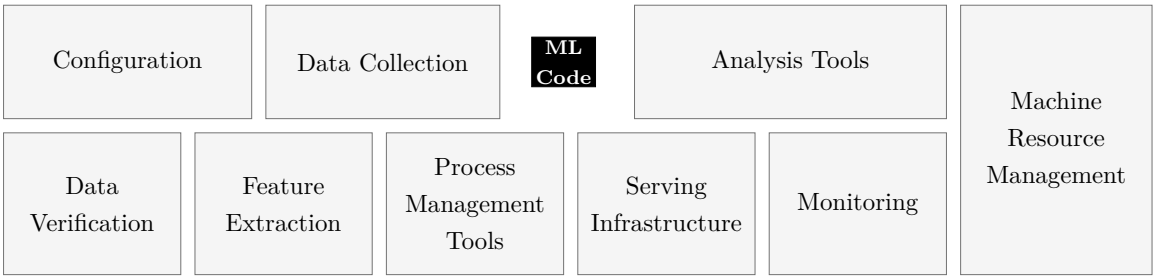


Figure 2.3: Only a small fraction of a real-world ML system is ML code. Adapted from Sculley et al. [13].

In a large case study at Microsoft, Amershi et al. [8] describe the integration of ML into software engineering processes as a nine-stage workflow, comprising model requirements, data collection, data cleaning, data labeling, feature engineering, model training, model evaluation, model deployment, and model monitoring. Amershi et al. [8] also identify three fundamental differences of ML-based development: i) discovering, managing, and versioning data is more complex than for code, ii) model customization and reuse require different skills than typical software engineering, and iii) ML components are entangled in non-modular ways. Subsequent surveys confirm that issues arise at every stage of the deployment lifecycle, from data management through model training and verification to serving, updating, and governance [35, 36], and that large-scale industrial ML systems face recurring challenges in adaptability, scalability, safety, and privacy [37, 38].

A large number of ML projects fail when moving ML prototypes and ML products, with many proofs of concept never progressing as far as production, since data scientists still manage large parts of the ML workflow manually [1, 14]. Precisely this gap motivates the transfer of DevOps principles to ML, which has crystallized into the MLOps paradigm.

Machine Learning Operations (MLOps) transfers the DevOps philosophy of automating and operating software to the specifics of ML systems. Based on a mixed-method study combining a literature review, a tool review, and expert interviews, Kreuzberger et al. [1] define MLOps as "a paradigm, including aspects like best practices, sets of concepts, as well as a development culture when it comes to the end-to-end conceptualization, implementation, monitoring, deployment, and scalability of machine learning products." As illustrated in Figure 2.4, MLOps is positioned at the intersection of machine learning, software engineering (especially DevOps), and data engineering, and aims at productionizing ML systems by bridging the gap between development (Dev) and operations (Ops) [1].

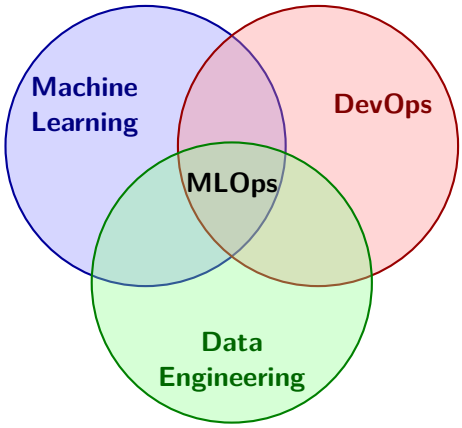


Figure 2.4: MLOps at the intersection of machine learning, software engineering, and data engineering (adapted from [1]).

Related conceptualizations describe MLOps as the standardization and streamlining of ML lifecycle management [39], or as the discipline of delivering ML applications through faster experimentation cycles and faster, quality-assured deployment [5, 40].

2.3.1 Principles

Kreuzberger et al. [1] identify nine principles required to realize MLOps, which directly extend DevOps practices with ML-specific concerns:

P1 CI/CD automation. Automated build, test, delivery, and deployment steps with fast feedback to developers. CI/CD puts the ideas of DevOps into practice [1, 41].

P2 Workflow orchestration. Coordination of the tasks of an ML pipeline according to directed acyclic graphs (DAGs) that capture task dependencies and execution order.

P3 Reproducibility. The ability to reproduce an ML experiment and obtain the same results.

P4 Versioning. Versioning of data, model, and code to enable reproducibility and traceability for compliance and auditing.

P5 Collaboration. Technical and cultural means to collaborate on data, models, and code, reducing domain silos across roles.

P6 Continuous ML training & evaluation. Periodic retraining of models on new data, enabled by monitoring, feedback loops, and an automated pipeline, always including an evaluation run.

P7 ML metadata tracking/logging. Tracking of metadata (parameters, metrics, lineage of data, and code) for every training job to ensure full traceability of experiments.

P8 Continuous monitoring. Periodic assessment of data, model, code, infrastructure, and serving performance to detect errors or changes affecting product quality.

P9 Feedback loops. Multiple loops integrating quality-assessment insights back into development, e.g., from monitoring to the retraining trigger.

2.3.2 Technical components

These principles are instantiated by nine technical components of the ML systems design [1]: a *CI/CD component* (C1, e.g., Jenkins, GitHub Actions) realizing build, test, delivery, and deployment, a *source code repository* (C2, e.g., GitHub, GitLab) versioning training, inference, and application code, a *workflow orchestration component* (C3, e.g., Apache Airflow, Kubeflow Pipelines) executing pipeline tasks as DAGs, a *feature store system* (C4, e.g., Feast) with an offline database for experimentation and an online database for low-latency serving, a *model training infrastructure* (C5, e.g., Kubernetes-managed, scalable compute resources), a *model registry* (C6, e.g., MLflow) storing trained models with their metadata, *ML metadata stores* (C7) tracking metadata of pipeline tasks and training jobs, a *model serving component* (C8) providing online or batch inference, typically containerized behind a REST API, and a *monitoring component* (C9, e.g., Prometheus/Grafana) continuously observing model serving performance and infrastructure. Comparable component inventories are reported in tool reviews and pipeline studies [5, 42, 11, 43].

2.3.3 Roles

MLOps is an interdisciplinary group process whose successful implementation depends on the interplay of several roles [1]: the *business stakeholder* (R1) defines the business goal, the *solution architect* (R2) designs the architecture and selects technologies, the *data scientist* (R3) translates the business problem into an ML problem and performs model engineering, the *data engineer* (R4) builds and manages data and feature engineering pipelines, the *software engineer* (R5) applies software engineering best practices to turn the ML solution into a well-engineered product, the *DevOps engineer* (R6) ensures CI/CD automation, deployment, and monitoring, and the *ML/MLOps engineer* (R7) combines cross-domain knowledge of all these roles to build and operate the ML infrastructure and the automated pipelines. The required combination of skills is scarce in practice, and the lack of such experts, as well as the prevailing model-centric mindset of data science teams, are recognized as adoption obstacles [1, 14].

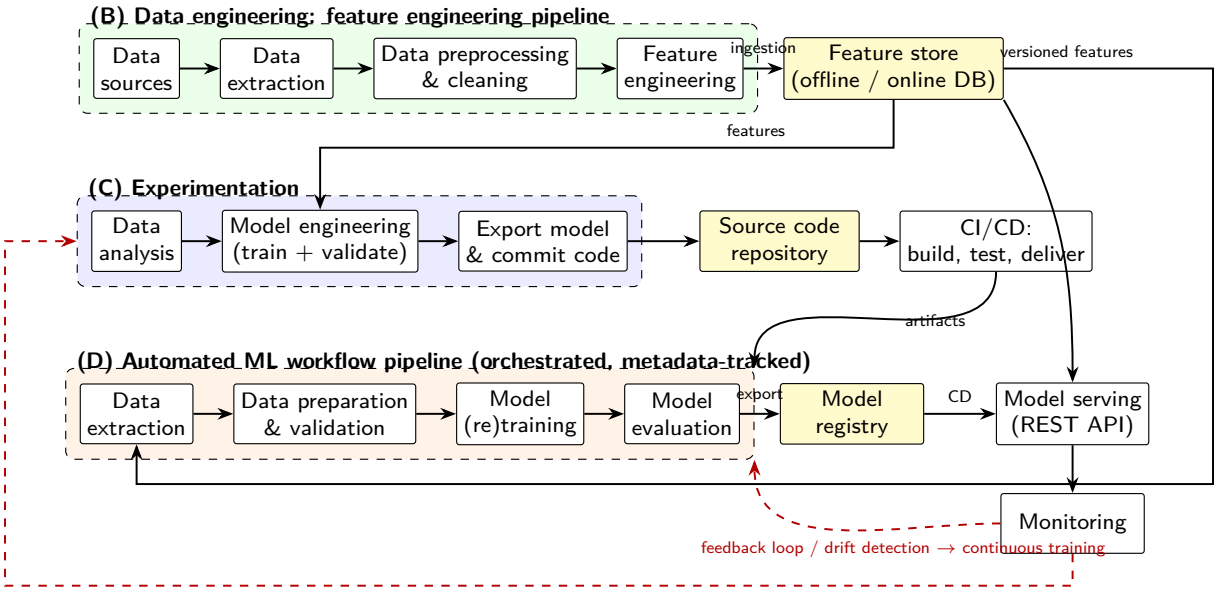


Figure 2.5: Simplified end-to-end MLOps architecture and workflow (adapted from [1]).

2.3.4 Architecture and workflow

Based on these principles, components, and roles, Kreuzberger et al. [1] derive a technology-agnostic, end-to-end MLOps architecture, a simplified version of which is shown in Figure 2.5. The workflow comprises four stages. In the *product initiation* stage (A), the business stakeholder identifies a business problem that can be solved with ML, the solution architect defines the system design, and the data scientist defines the corresponding ML problem. In the *data engineering* stage (B), a feature engineering pipeline extracts, preprocesses, and transforms raw data and ingests the resulting features into the offline and online databases of the feature store. In the *experimentation* stage (C), the data scientist analyzes the data, performs model engineering (iterative training and validation to identify the best-performing algorithm and hyperparameters), and commits model and code to the repository, upon which the CI/CD component builds, tests, and delivers versioned pipeline artifacts. Finally, the *automated ML workflow pipeline* (D), managed by the orchestration component, automatically extracts versioned features, prepares and validates data, retrains and evaluates the model, and pushes it to the model registry. At the same time, metadata is logged for every task. Once promoted to production, the model is deployed by the continuous deployment pipeline to the serving component, which answers online or batch prediction requests. The monitoring component observes serving performance in real time, and feedback loops propagate observed degradations, e.g., concept drift, when the statistical properties of input data change over time, back to the experimentation zone, the data engineering zone, or directly to a scheduler that triggers retraining (continuous

training) [1].

2.3.5 Maturity and open challenges.

The transition towards such fully automated setups is gradual. John et al. [40] propose a maturity model in which organizations evolve from manual, script-driven ML workflows over automated training pipelines to fully automated CI/CD-integrated retraining and deployment. A widely used industrial reference, the Google Cloud MLOps maturity model [44], makes this progression concrete through three levels of automation. At Level 0, building and deploying models are entirely manual and depend on a script- or model-based hand-off between data science and operations. Level 1 automates the ML pipeline to enable continuous training on new-data triggers. Level 2 adds a full CI/CD infrastructure that integrates the model and its training pipeline, supporting rapid and reliable updates with little manual intervention [44, 45]. We use this progression as an analytical reference to distinguish teams that remain dependent on manual handoffs from those with more integrated production practices. Steidl et al. [11] show that published continuous-development pipelines for AI models vary considerably in their automation, testing, and monitoring stages. Despite its growing adoption, MLOps still faces open challenges, which Kreuzberger et al. [1] organize into three categories: *organizational* challenges, such as the required culture shift from model-driven data science to product-oriented engineering, the shortage of skilled architects, data engineers, and ML engineers, and the need for multidisciplinary, non-siloed teams, *ML system* challenges, in particular designing for fluctuating demand and voluminous, varying data, which complicates infrastructure sizing and demands high scalability, and *operational* challenges, including the robust automation of heterogeneous hardware/software stacks, governance and versioning of the many artifacts produced by repeated retraining, and root-cause analysis when failures span the ML infrastructure and the surrounding software. Similar technical, organizational, and operational issues are reported in practitioner studies and surveys [14, 5, 39, 37].

MLOps extends the cultural and technical practices that made DevOps successful, CI/CD, automation, collaboration, and continuous monitoring, with ML-specific principles such as data/model versioning, experiment tracking, continuous training, and drift-aware feedback loops. This background frames the remainder of this work, which builds upon the MLOps principles, components, and architecture described above.

2.4 Challenges of Implementing MLOps

The literature consistently reports that implementing MLOps is difficult along three interrelated dimensions, which Kreuzberger et al. [1] categorize as organizational, ML-

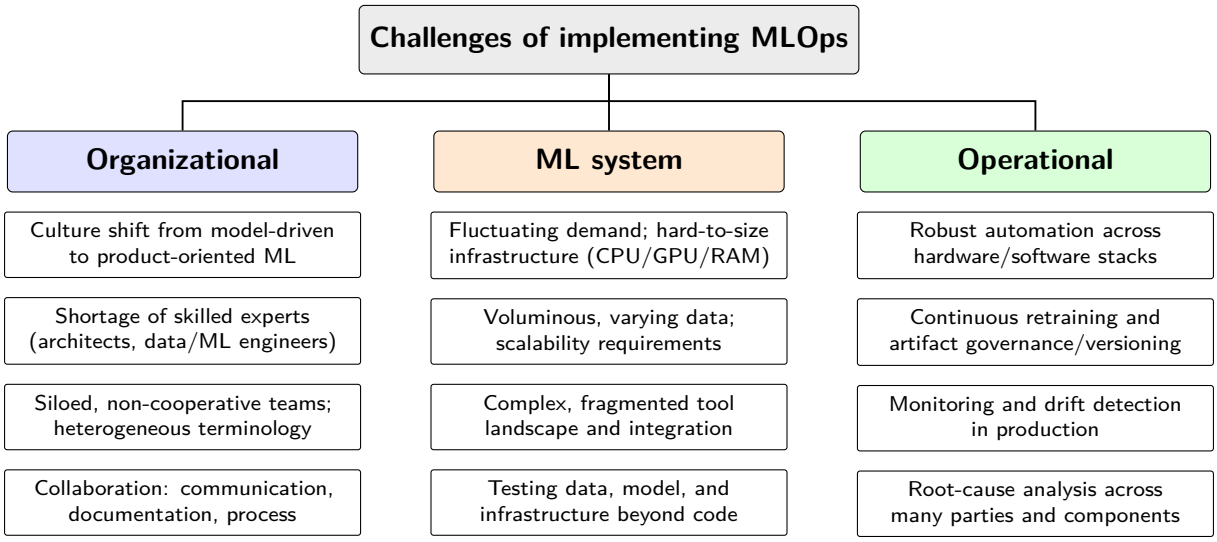


Figure 2.6: Taxonomy of MLOps implementation challenges (own representation, based on [1, 46, 35, 5]).

system, and operational challenges. Figure 2.6 summarizes this taxonomy, enriched with findings from empirical studies of ML practice.

2.4.1 Organizational challenges

The dominant obstacle is cultural. Successfully developing and operating ML products requires a shift away from model-driven data science towards a product-oriented engineering discipline [1]. Mäkinen et al. [14] surveyed 331 ML practitioners and found that a large share of teams are still at the stage of getting first models into production at all, and that data scientists' daily work is dominated by manual, infrastructure-related plumbing for which MLOps practices are intended but not yet established. Implementing MLOps also demands a combination of skills, spanning data science, data engineering, software engineering, and operations, that is scarce on the labor market. The lack of experienced architects, data engineers, ML engineers, and DevOps engineers is repeatedly named as a bottleneck, aggravated by the fact that MLOps is typically not part of data science education [1]. Finally, MLOps is inherently a group process. As a result, data scientists alone cannot achieve its goals, yet teams often work in silos with different knowledge levels and specialized terminology that hamper communication [1]. Nahar et al. [46], interviewing 45 practitioners from 28 organizations, locate that the most severe collaboration problems addressed by MLOps are unclear responsibility boundaries between model development and operations, insufficient documentation of models and data, and mismatched processes between exploratory data science work and disciplined engineering cadences. Wan et al. [47]

confirm, based on a large survey, that ML development differs from traditional software development in requirements, design, testing, and processes, implying that established DevOps routines cannot simply be copied onto ML teams.

2.4.2 ML-system challenges

A key challenge is designing for fluctuating demand and varying data volumes, which complicates estimating the necessary infrastructure resources (CPU, RAM, GPU) and necessitates highly scalable, often distributed infrastructure [1, 37]. In this context, practitioners must compose feature stores, orchestrators, registries, serving, and monitoring stacks from a rapidly evolving market of open-source and commercial offerings, and integrating these heterogeneous components into one coherent pipeline is itself a recognized challenge [5, 42, 11]. Moreover, quality assurance must extend beyond code to data and models. Production readiness requires systematic tests for features and data distributions, model development, and serving infrastructure, together with continuous monitoring, which most teams only partially implement [33, 35].

2.4.3 Operational challenges

Once an ML product is live, operating it manually is impractical due to the interplay of different software and hardware stacks, so robust automation is required [1]. The constant inflow of new data forces repeated retraining, which yields a large number of artifacts, models, datasets, features, and metadata that demand strong governance and versioning to preserve robustness and reproducibility [1, 41]. Monitoring must detect not only infrastructure failures but also silent degradations such as concept drift, and support requests are hard to resolve because failures may originate in any combination of ML infrastructure, data, and application software [1, 35]. Granlund et al. [48] report from two real-world healthcare cases that, in multi-organization settings, integration across organizational boundaries, splitting MLOps responsibilities between independent parties, and regulatory constraints on data and models add a layer of complexity that single-company MLOps blueprints do not cover.

2.4.4 Software Requirements

Requirements engineering (RE) is the software engineering discipline concerned with eliciting, specifying, validating, and managing the needs and constraints a system must satisfy. For ML-enabled systems, RE is recognized as one of the most strongly affected activities. Practitioners find that eliciting and specifying requirements for ML systems is fundamentally harder than for traditional software, as the system behavior is learned

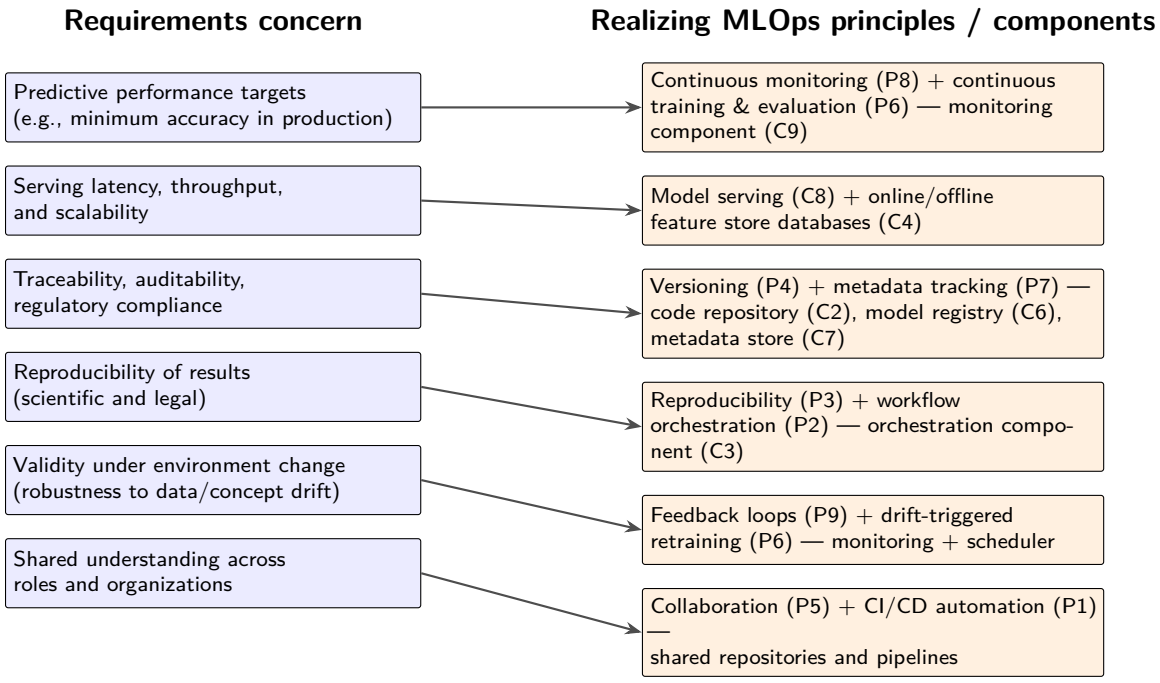


Figure 2.7: Requirements concerns of ML-enabled systems mapped to MLOps principles and components (own representation).

from data instead of programmed against a specification [47, 8]. Vogelsang and Borg [49] argue that requirements engineers working on ML must understand ML performance measures to state meaningful functional requirements. They must also address new quality concerns, such as explainability and freedom from discrimination. They must extend RE to data, because requirements increasingly concern data quantity, quality, distribution, and provenance. Systematic mappings of the emerging RE4AI field confirm both its growth and its immaturity. Existing approaches concentrate on eliciting requirements for model performance while largely neglecting requirements for data pipelines, monitoring, and operations, precisely the concerns MLOps addresses, and validated notations or tools for specifying them are scarce [50, 51]. Non-functional requirements (NFRs) pose specific challenges because traditional NFR catalogs do not align well with ML systems. New qualities (fairness, transparency, retrainability) appear, and NFRs may be defined in relation to the model, the data, or the entire system, and the appropriate scope for measuring each is often unclear [52, 53].

In MLOps context, requirements determine how the MLOps reference architecture is instantiated. Figure 2.7 maps recurrent requirements concerns of ML-enabled systems to the MLOps principles and components that realize them. A serving-latency requirement, for example, decides whether predictions are computed online from the low-latency online feature store or in batch from the offline store [1]. An accuracy-in-production requirement is meaningless without the monitoring component that measures it. Auditability and

regulatory requirements, prominent in domains such as healthcare, translate directly into the versioning of data, model, and code and into ML metadata tracking, and may even shape how pipeline responsibilities are split across organizations [48, 1]. In this sense, large parts of the MLOps stack can be read as infrastructure for satisfying non-functional requirements that traditional deployment processes leave implicit.

In the other direction, MLOps changes the nature of requirements themselves. For a learned component, the central functional requirement, that the model generalizes adequately to operational data, cannot be conclusively verified before deployment. It can only be checked continuously while the system runs. The MLOps principles of continuous monitoring, continuous evaluation, and feedback loops [1] effectively shift requirements validation from a design-time to a runtime activity: monitored performance thresholds serve as machine-checked acceptance criteria. A detected concept drift is the system observing that the environment no longer satisfies the assumptions under which its requirements were validated, triggering retraining to restore compliance [1, 13]. This continuous treatment of requirements is a natural extension of the continuous software engineering vision, in which planning, integration, delivery, and verification become ongoing activities [26].

Finally, empirical evidence suggests that many MLOps implementation failures are, at their core, requirements failures. Lewis et al. [54] characterize *ML mismatch* as problems arising from incorrect assumptions made about ML components by the different perspectives involved (data science, software engineering, operations), and trace mismatches in deployed systems to undocumented expectations about training data, operational data distributions, computing resources, and monitoring needs, in essence, unstated requirements that were never elicited or shared. Their proposed remedy, machine-readable descriptors of ML components agreed upon by all stakeholders, is a requirements specification practice for the MLOps pipeline. Nahar et al. [46] likewise identify requirements-related collaboration points, such as negotiating model targets with product owners and documenting data expectations between teams, as the places where ML projects most need process maturity. These findings close the loop with the organizational challenges discussed in Section 2.4: implementing MLOps successfully requires not only pipelines and tools, but explicit, shared, and continuously validated requirements over data, models, and operations [54, 46, 50].

In summary, the challenges of implementing MLOps are organizational, system-related, and operational. However, they converge on a common theme: the assumptions and constraints governing an ML product must be made explicit and kept verifiable throughout its lifecycle. Software requirements provide exactly this function. MLOps, in turn, supplies the technical machinery, versioning, metadata tracking, monitoring, and feedback loops that make ML-specific requirements specifiable, traceable, and continuously checkable in production.

2.5 Related Work

Several studies have sought to consolidate the practices that make MLOps work in production. Alves et al. [2] synthesize a catalog of MLOps practices spanning the ML lifecycle, from data management to deployment and monitoring. Their work stresses the need to adapt traditional software engineering practices to the uncertainties of data, models, and code. We adopt this catalog as the deductive basis for identifying the practices reported in this study, referenced throughout as P-xx. Zarour et al. [55] and Eken et al. [4] add complementary syntheses of pipeline-automation techniques, continuous-training principles, and requirements for consistent artifact management. Despite these efforts, the field remains fragmented, with no single agreed-upon reference for how MLOps should be implemented in practice.

A second and more recent strand looks past pipeline architecture to the socio-technical friction that most often prevents models from reaching production. Amrit and Narayanappa [16] analyze expert interviews with grounded-theory coding and organize the challenges of MLOps adoption into four aggregate dimensions: organizational, technical, operational, and business. We adopt this categorization to structure our own findings (Chapter 4), complementing the three-category view of Kreuzberger et al. [1]. Mailach and Siegmund [56] identify seventeen socio-technical anti-patterns through a qualitative analysis of practitioner talks, including painful model handovers, redundant development, and a "leadership vacuum" that drives hype-oriented product decisions. Nahar et al. [46] interview practitioners across dozens of organizations and locate the most severe difficulties at the collaboration points between data science and engineering: unclear responsibility boundaries, insufficient documentation, and mismatched process cadences. A recurring diagnosis across these studies is that the cultural clash between research-oriented data scientists and process-oriented engineers produces a "wall of confusion" that only cross-functional restructuring resolves [8, 46].

This diagnosis connects to the DevOps team-structure literature discussed in Section 2.2. Because an organization's architecture tends to mirror its communication structure [29], the division of ML, data, and operations responsibilities across teams shapes and often constrains how ML systems can be built and delivered. The taxonomies of López-Fernández et al. [31] and Díaz et al. [21] make this concrete by linking team topology to software-delivery performance, from siloed departments through cross-functional and platform-supported teams. We return to this connection when interpreting the coordination challenges observed in our cases.

These challenges are not uniform across organizations. Small teams and projects in early phases face disproportionate difficulty adopting MLOps and report lower success rates, because of the high infrastructure cost and specialized expertise required to build and

maintain automated pipelines [4, 57]. Such teams depend more on manual workflows, lack the architectural guardrails available in larger enterprises, and are prone to underestimating the operational cost of ML [16]. Most MLOps frameworks and empirical studies, however, assume abundant resources and dedicated MLOps teams, which leaves the resource-constrained, first-time adopter comparatively understudied.

Empirically, MLOps has been examined through case studies, though rarely in constrained settings. Faubel and Schmid [58] report a multiple-case study of Industry 4.0 organizations and show that industrial MLOps must balance the pull toward standardized platforms against high environmental variability and human-in-the-loop requirements. Granlund et al. [48] show, in two healthcare cases, that multi-organization settings add integration, responsibility-splitting, and regulatory constraints that single-company blueprints do not address. Sambasivan et al. [59] document how undervalued "data work" produces downstream data cascades, a mechanism we observe directly in our data-dependency findings.

Our study differs from this prior work in two respects. Methodologically, existing qualitative accounts of MLOps friction rely largely on retrospective, self-reported interview data, and Amrit and Narayanappa [16] note the scarcity of evidence on how practices operate day to day. We instead triangulate participant observation and the live analysis of organizational artifacts (repositories, issue trackers, meeting transcripts) with semi-structured interviews, which lets us capture practices as they unfold. Contextually, prior work centers on large, resource-rich organizations, whereas we study two resource-constrained teams adopting ML for the first time in Brazilian government-adjacent and academic settings, where infrastructure, budget, and data access are externally constrained. In that setting, we examine which challenges dominate and which practices enable deployment.

Chapter 3

Methodology

3.1 Case Selection

We selected two cases to investigate these phenomena. The first involves the development of ML components for Brasil Participativo, a civic participation platform. The second case encompasses the end-to-end development of AdsRay, a tool designed to detect news with disinformation in Brazilian newspapers. These cases were chosen because both focused on delivering ML-based products in real organizational environments and involved multidisciplinary teams working on the development, deployment, and maintenance of machine learning models in production. Furthermore, both projects have ongoing MLOps initiatives and face similar challenges in scaling ML-enabled solutions, which makes them relevant for investigating lifecycle practices, barriers, and interactions among data, modeling, engineering/MLOps, and organizational coordination.

3.2 Case A: Brasil Participativo

Brasil Participativo is the Brazilian federal government's digital platform for social participation, developed through a collaboration between the University of Brasília (UnB) and the Office of the Presidency of the Republic under the National Secretariat for Social Participation. Built on Decidim, an open-source participation framework, it brings mechanisms such as public consultations and participatory plans into a single environment. Its first process was the 2024–2027 Participatory Pluriannual Plan (PPA), the largest digital participatory process in the country's history, and over its first two years the platform registered approximately 1.6 million users and more than 42,000 text-based proposals. The setting shapes this study in three ways. It is government-adjacent, and therefore subject to public-sector infrastructure and data-governance constraints. It is built by a three-sector collaboration, government, academia, and industry, with different workflows

and cultures, and it runs at a national scale that makes it a real, non-deterministic production environment.

The platform was initially structured around three core teams: **Product Engineering**, **Design**, and **Development**. The Product Engineering team was responsible for requirements elicitation and defining the platform's functional and participatory needs. The Design team focused on user experience and interface design, ensuring accessibility and usability across the platform's participatory features. The Development team was responsible for implementing the functionalities specified and prototyped by the other teams. New challenges emerged in analyzing and moderating large volumes of citizen-generated content as platforms evolved and their user bases expanded. In this context, the project identified the need to incorporate AI-based capabilities to support tasks that would be difficult to perform with traditional software approaches or manual human analysis alone. One of the first ML-driven initiatives on the platform involved automatically classifying hate speech in user contributions. Given the platform's participatory and socially sensitive context, the growing volume of interactions necessitated the development of mechanisms to detect and moderate harmful content at scale. This demand motivated the creation and structuring of the **Data Science** team within the project.

The platform's participatory scope and institutional context underscored the need for a multidisciplinary team to develop it, including software engineers, designers, researchers, public policy specialists, and Data Science practitioners. The project involves academic researchers, government stakeholders, and technical contributors, which demands continuous alignment between technical implementation, requirements, and institutional constraints.

In this context, the contributors of the platform were organized into six teams responsible for different aspects of the system's development and operation: **Product Engineering**, **Development**, **Design**, **Data Engineering**, **Infrastructure**, and **Data Science**. Figure 3.1 presents the team's configuration. In this configuration, Product Engineering coordinates the build teams and owns requirements and prioritization, Data Engineering hands off processed data to Data Science, and Infrastructure provides the shared platform on which the build teams operate and depends on the external state IT provider (Dataprev) for cloud services and data access.

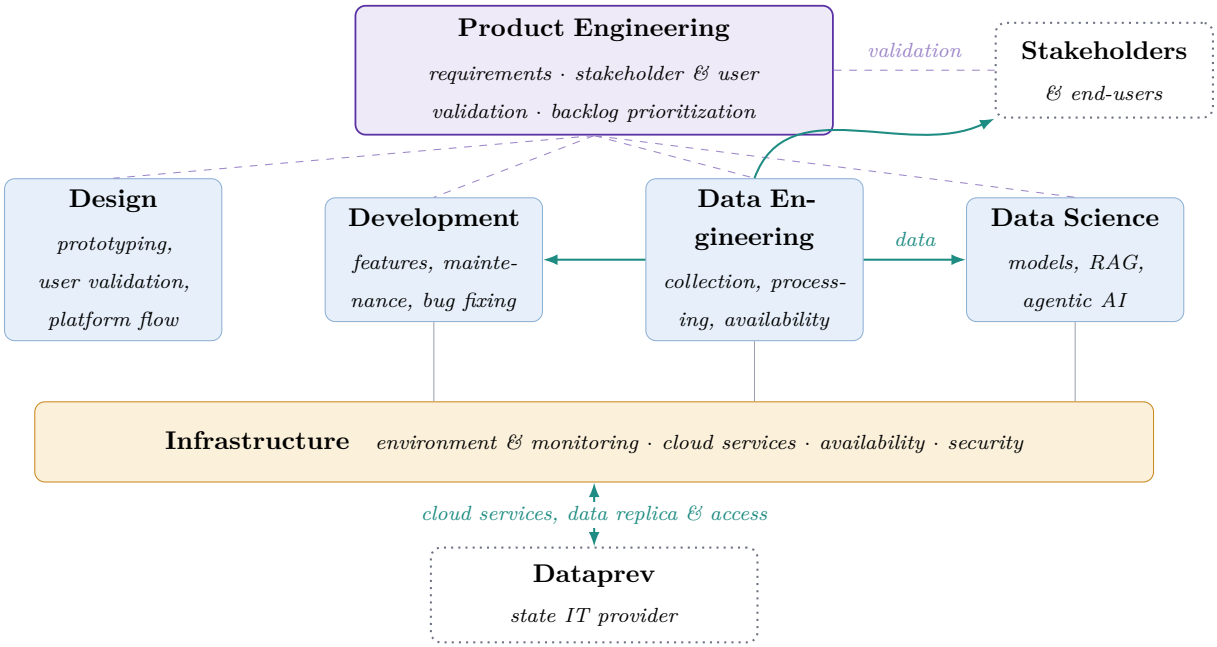


Figure 3.1: Team structure of Case A (Brasil Participativo).

Although the Data Science team was first created for content moderation, its work grew into three products, each with its own cross-team dependencies (Figure 3.2).

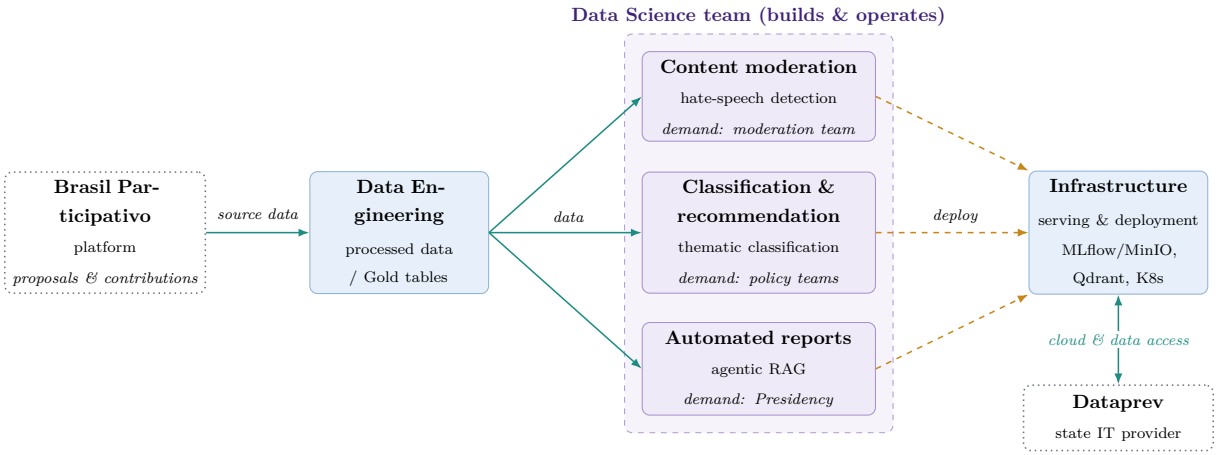


Figure 3.2: Case A ML products and their cross-team dependencies.

Content moderation. The first product, and the one that started the Data Science front, detects hate speech in proposals and contributions so that abusive content can be flagged without a moderator reading everything. It fine-tunes DeBERTinha, a small DeBERTa model pre-trained for Portuguese, on the HateBR hate-speech corpus and evaluates it on platform data, and it is served through MLflow and MinIO on the

laboratory's infrastructure. The design is supervised. The model triages the obvious volume, a moderator confirms or overturns each decision in an admin panel, and those corrections are logged to feed retraining.

Proposal classification and recommendation. This product organizes the proposal corpus by theme and recommends related proposals across processes. Classification uses a BERTopic topic-modeling pipeline with BERTimbau embeddings, combining an unsupervised pass with semi-supervised passes seeded from the government's controlled vocabulary (the VCGE), so that the resulting themes align with the categories public systems already use. Large language models assist the labeling and topic naming. The pipeline was trained on about ten thousand proposals drawn from three participatory processes and covers dataset splitting, hyperparameter optimization, training, deployment, and an inference API. The recommendation feature suggests related proposals to a reader, drawing on the categories produced by the classification model.

Automated analytical reports. This product generates a report for each participatory process, with an executive summary, participation indicators, and a trace of how citizen proposals were incorporated into policy. It is an end-to-end agentic RAG system built on Haystack and Hayhooks, with Qdrant as the vector store and Ollama serving the embedding models, and it is containerized for Kubernetes. An automated ingestion pipeline standardizes the platform's data, the agent retrieves the relevant content and grounds the generated text in it, and a manager triggers a report and receives it by email.

On the Google Cloud MLOps maturity model [44], these three products sit at Level 0. Their training and deployment steps are largely manual, with no automated continuous training on fresh data, the capability that defines Level 1, nor the CI/CD-integrated retraining of Level 2.

Figure 3.3 summarizes the tooling the Data Science team assembled across the three products, grouped by function.

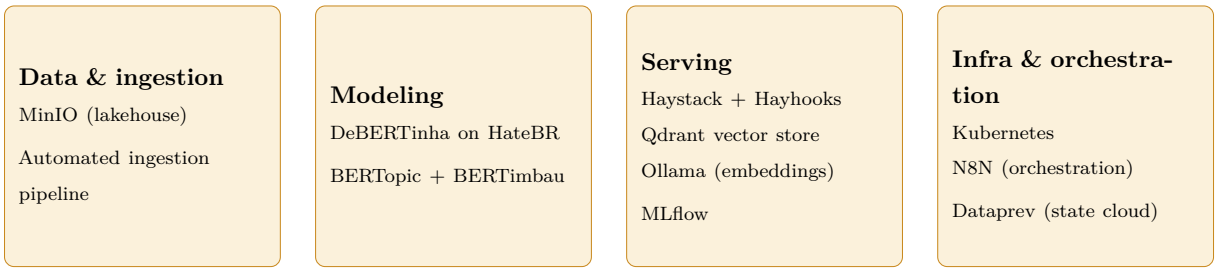


Figure 3.3: Case A tooling, grouped by function.

3.3 Case B: AdsRay

AdsRay is an open-source disinformation-detection tool developed by the **Lab Livre** research laboratory at the University of Brasília (UnB) together with the nonprofit **Sleeping Giants Brasil**. Sleeping Giants Brasil works against disinformation through its financing. It identifies news outlets that recurrently publish disinformative content and alerts the advertisers whose ads, placed through networks such as Google Ads, fund those outlets, so that this revenue can be withdrawn. Over several years the organization built a manual methodology for this work, mapping ninety news portals with a history of disinformation and defining classification criteria organized into five thematic axes: gender and sexuality, ethnic-racial relations, collective health, democratic stability, and the socio-environmental agenda, but this manual analysis could not scale. Classifying a set of only one thousand articles from the mapped portals took the organization's analysts around six months, which made it impossible to keep pace with daily news production. AdsRay automates this methodology so that the same expert-defined criteria can be applied continuously and at scale, with human review retained over the results.

The tool was developed in an interdisciplinary arrangement.

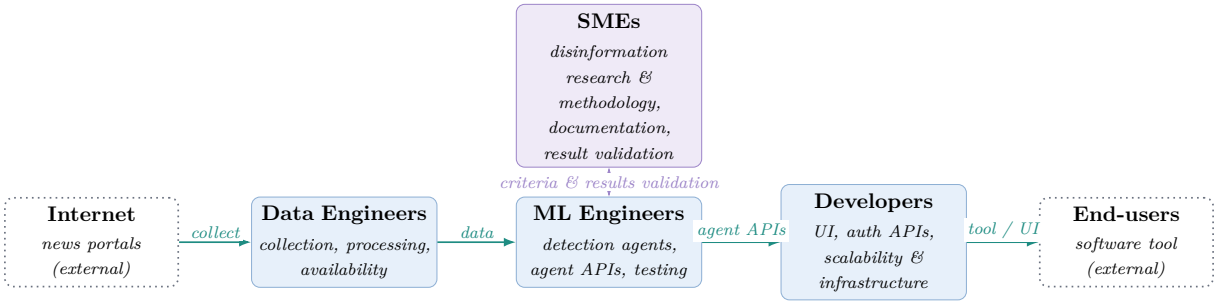


Figure 3.4: Team structure of Case B (AdsRay).

In Figure 3.4, we present the team responsible for developing the platform into four areas: **Subject Matter Specialists (SMEs)**, **Developers**, **Data Engineers**, and **ML Engineers**.

- **SMEs**: This team is the domain specialist. Its main task is to research the problem and the methodologies associated with the dissemination of disinformation, and supply the domain criteria (thematic-axis ground truth) and validate the tool's results.
- **Developers:** This team focuses on implementing the software tool. It develops the user interface and the necessary authentication APIs, as well as the infrastructure required for scalability.

- **Data Engineers:** This team is responsible for data collection, processing, and availability. Its main responsibility is to gather data available on the Internet, store it, and make it available to the ML team.

- **ML Engineers:** This team is responsible for researching and developing disinformation detection agents. Their responsibility is to test AI agents to detect disinformation, develop the APIs for these agents, and make them available for integration with the software solution.

The project was developed in an interdisciplinary, collaborative environment, with ML Engineers, Software Engineers, Data Engineers, and SMEs interacting continuously throughout development. Unlike Case A, where each team had a specific responsibility, in this Case overlapping responsibilities were common among teams, particularly in infrastructure provisioning, data management, and deployment-related activities. Also, interaction among members was more frequent because the team was small and easier to manage.

AdsRay implements the organization's methodology as an automated pipeline structured in two stages.

Extraction and processing. Apache Airflow orchestrates the flow. On a recurring schedule, an extraction stage collects articles from the mapped portals, with dedicated scrapers built for a growing subset of them (46 at the time of this study), and stores each one as structured JSON in MinIO, organized as a lakehouse partitioned by portal and date. Airflow then triggers the processing stage, which sends the stored articles in batches to a Retrieval-Augmented Generation (RAG) service built on Haystack. The enriched output feeds a web interface that presents each classification together with its justification and offers filters by portal and thematic axis. In preliminary runs the pipeline processed on the order of three thousand articles per day.

Classification. The classification stage applies RAG in two retrieval steps over a Qdrant vector store that holds two kinds of documents, previously classified articles and the journalistic disinformation criteria, both labeled by thematic axis. Text is embedded by a local model served through Ollama, which keeps the data in-house and avoids external API costs. For an incoming article, the pipeline splits it into overlapping chunks and uses semantic similarity against the historical articles to infer which thematic axes apply. It then retrieves only the disinformation criteria for those axes and passes them to the language model, which decides whether the article is disinformative and cites the passage and the criterion it violates. Restricting the retrieved context to the relevant axes keeps the prompt focused and produces auditable justifications. From these decisions each portal

receives a disinformation index per thematic axis, and portals above a defined threshold enter a block list that the organization uses to engage advertisers.
Figure 3.5 summarizes this deliverable end to end, from portal collection to the advertiser block list.

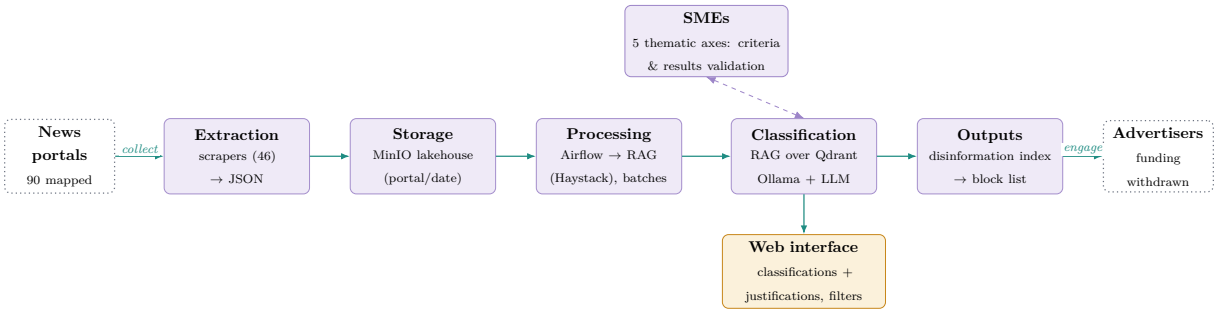


Figure 3.5: Case B deliverable: the disinformation-detection pipeline, from portal collection to the advertiser block list.

On the Google Cloud MLOps maturity model [44], this pipeline sits at Level 0. Although Apache Airflow orchestrates extraction and enrichment on a fixed cadence, model building and updates remain manual, without the automated continuous training that defines Level 1 or the CI/CD-integrated retraining and drift monitoring of Level 2.
The tooling behind these stages is summarized in Figure 3.6, grouped by function.

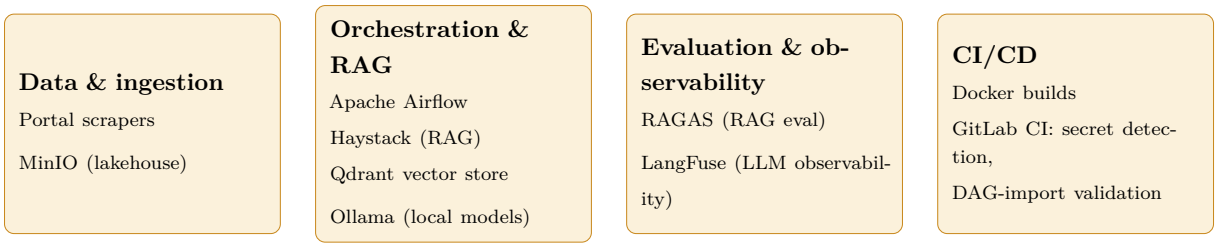


Figure 3.6: Case B tooling, grouped by function.

3.4 Positioning the cases on the MLOps maturity model

The two cases are first-time, resource-constrained adopters, but they operate their ML with different degrees of tooling. To compare them on a common basis, we position each against the Google Cloud MLOps maturity model [44] (Levels 0–2), introduced in

chapter 2. Table 3.1 summarizes the characterization, and the per-case descriptions above give the underlying detail.

Table 3.1: The two cases against the capabilities of the Google Cloud MLOps maturity model [44].

Capability	**Case A**	**Case B**
Level 1 — ML pipeline automation		
Continuous training (CT)	No	No
Experimental-operational symmetry	No	Partial
Modularized code for components and pipelines	Partial	Partial
Continuous delivery of models	No	No
Data and model validation	Partial	Partial
Feature store	No	No
Metadata management	Partial	Partial
ML pipeline triggers	No	Partial
Level 2 — CI/CD pipeline automation		
Pipeline continuous integration	No	Partial
Pipeline continuous delivery	No	Partial
Automated triggering	No	Partial
Model continuous delivery	No	No
Monitoring	No	Partial
Assessed level	**Level 0**	**Level 0**

Both cases sit at Level 0 of the model: model building, training, and deployment are manual and script-driven, and neither automates continuous training on fresh data, the capability that defines Level 1, nor the CI/CD pipeline automation of Level 2. Their monitoring is, in both, absent or partial. What differs is the operational tooling wrapped around that manual core. Case A's three ML features are split across specialized teams connected by manual handoffs, with an N8N workaround around the platform's release path, whereas Case B invests more in pipeline orchestration (Airflow), continuous integration (containerized builds, secret detection, and pipeline-import validation), model evaluation (RAGAS), and observability (LangFuse) on a single stack. Placing both at the same maturity level, reached by different operational routes, is what makes the comparison informative: they face the same class of first-time, resource-constrained operationalization problem, so their common challenges and the practices each adopts can be read against a single reference.

3.5 Study Design

Figure 3.7 summarizes our research design. We conducted an exploratory qualitative multi-case study, following the guidelines proposed by Runeson and Höst [61], to investigate the factors that define successful machine learning adoption.

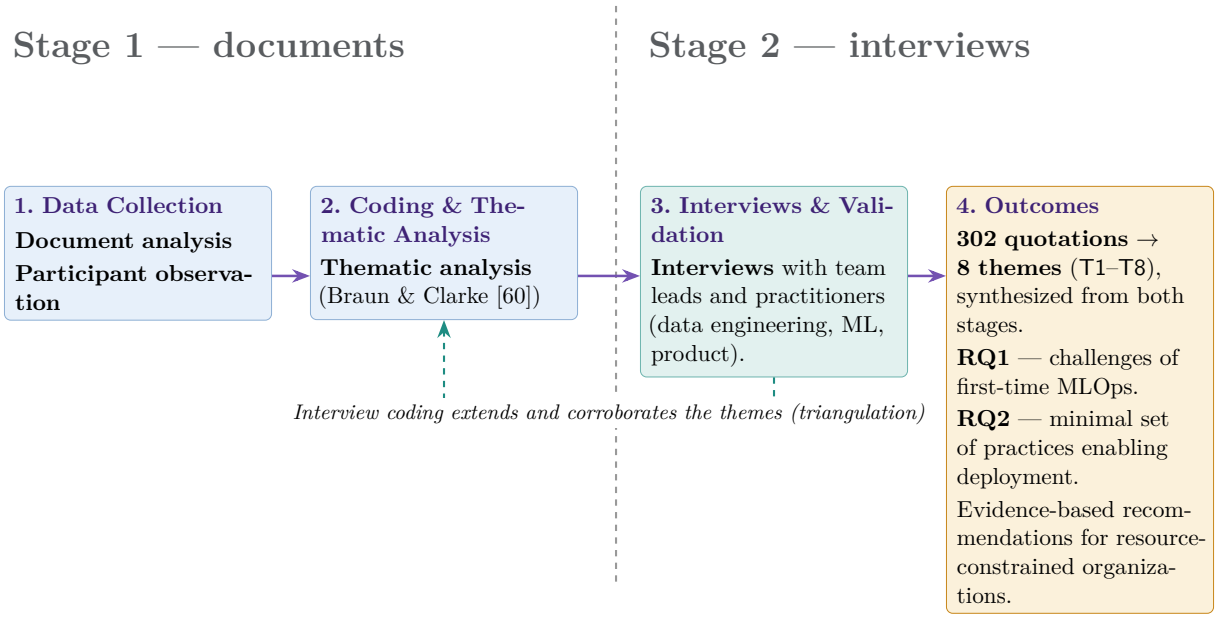


Figure 3.7: Research design: a two-stage, multi-case study.

This approach is well-suited to the research problem because it supports a holistic understanding of the barriers that hinder the transition from development to production in first-time adopter organizations (RQ1), as well as the technical, organizational, and socio-technical practices that practitioners adopt to operationalize ML systems (RQ2). The combination of evidence from interviews, meeting transcriptions, and technical artifacts allows us to capture complementary perspectives on how data practices, modeling activities, software engineering processes, MLOps capabilities, and organizational coordination interact during ML operationalization.

3.5.1 Unit of Analysis

In each case, the unit of analysis is the operationalization of an ML component, meaning the technical practices, engineering decisions, organizational arrangements, collaborative mechanisms, and operational artifacts through which a component moves from experimentation to sustained production use. The design is embedded, in which Case A contributes three components (content moderation, proposal classification and recommendation, and automated reports) and Case B contributes one (the disinformation-classification pipeline). The analysis operates at the level of teams and their coordination, not individual performance.

3.5.2 Success Indicators

We define success at the entry level of ML operationalization, since both cases are first-time adopters working under resource constraints and we frame both at Level 0 of Google MLOps maturity model. A component counts as operationalized when it leaves the experimental environment and runs as a served, data-fed, and maintained part of the product. Table 3.2 lists the concrete indicators we looked for, assessed by triangulating interviews, internal documents, and operational artifacts.

Table 3.2: Indicators of an operationalized ML component.

Indicator	**What we looked for**
Data availability	A repeatable data source or pipeline feeds the component.
Served	The model is reachable through an API or a deployed service.
Shared infrastructure	The component runs on team or laboratory infrastructure and keeps working without its author being present.
Retrievable artifact	The trained model is stored and versioned so it can be redeployed or replaced.
Integrated into the product	Its output is consumed by the application or an end user.
Operable by the team	More than one person can run, redeploy, or fix it, so it does not depend on a single individual.

3.6 Data Collection

The study ran from April 2025 to July 2026. Over this period, we analyzed 67 technical artifacts (GitLab and Notion issues and technical design documents) and 16 meeting transcriptions, and we conducted nine semi-structured interviews with practitioners across the two cases. In Case A, we interviewed a subject-matter specialist, three Data Science engineers, and two Data Engineering technical leads. In Case B, a subject-matter specialist and two ML engineers. The interviews were conducted remotely, in Portuguese, and lasted around 45 minutes each. Each interview was recorded with the participant's verbal consent and transcribed. Excerpts quoted in this thesis were translated to English, with the original Portuguese retained in our analysis records.

3.6.1 Document Collection

In Stage 1, we collected internal documents that record influential practices, barriers, and coordination mechanisms across the cases, filtered by relevance to the ML lifecycle and categorized into three types. Table 3.3 summarizes these artifact types, what we examined in each, and why.

Table 3.3: Document sources analyzed, with what we examined in each and the rationale.

Artifact type	**What we examined**	**Rationale**
Management	Notion documents, GitLab issues and pull requests, with their comments	Capture task definition, project milestones, and the discussion around ML-product changes and delivery.
Technical	README files and architecture documentation	Record the intended design and technical structure of each ML component.
Collaborative	Team-meeting transcriptions and notes (recorded with Fireflies)	Surface coordination, handoffs, and decisions as the teams discussed them.

To assemble this corpus, we first mapped all the evidence available in each case. From the GitLab projects we collected the issues and pull requests related to the ML products, together with their comments, and the repositories' architecture documentation and README files. From Notion we collected the project documentation. We also added the team-meeting transcriptions recorded with the Fireflies notetaker. We consolidated every candidate item into a single spreadsheet and applied one exclusion criterion, discarding any evidence not related to ML development.

To make this step repeatable, we automated it. We connected a large language model (Anthropic, Claude Opus 4.8) to the GitLab and Notion Model Context Protocol (MCP) servers and encapsulated the retrieval in a purpose-built skill that queried each source, applied the relevance criterion, and exported the filtered corpus to the spreadsheet used for the analysis.

Table 3.4 summarizes the composition of the analyzed corpus by evidence source and case. The technical artifacts, drawn from the projects' GitLab and Notion workspaces, comprise GitLab and Notion issues, Notion documents, and GitLab technical documents. Alongside them, the corpus includes recorded meeting transcriptions and interviews.

Table 3.4: Composition of the analyzed corpus by evidence type and case.

Type	Case A	Case B	Total
GitLab issues	16	–	16
Notion issues	19	–	19
Notion documents	26	–	26
GitLab documents	–	6	6
Meeting transcriptions	16	–	16
Interviews	6	3	9
Total	**83**	**9**	**92**

3.6.2 Interviews

In Stage 2, we collected data through semi-structured interviews with practitioners (e.g., data scientists, data engineers, product managers, and product owners) to explore participants' experiences and perceptions and to enable comparability across interviews. The interviews were designed to elicit evidence about three aspects aligned with the research questions: (i) practices perceived as influential for ML adoption and productionization, (ii) barriers that hinder sustained operation, and (iii) interactions among technical, organizational, and coordination practices.

We developed the interview guide following the five-phase framework proposed by Kallio et al. [62]: (i) identifying the prerequisites for semi-structured interviews, (ii) retrieving and utilizing prior knowledge, (iii) formulating the preliminary guide, (iv) pilot testing, and (v) finalizing the interview protocol. This process strengthens the reliability and validity of the research instrument by grounding the questions in both theory and the practical context of the cases. We developed the interview protocol through the following phases:

- **(i)** We analyzed the research context in the cases and determined that semi-structured interviews are the most appropriate method for this study, given the exploratory nature of understanding lifecycle practices, barriers, and coordination mechanisms in ML adoption. This choice provides the necessary flexibility to capture complex perceptions while maintaining comparability across the selected cases.

- **(ii)** We built a foundation by combining established MLOps literature with my direct observations as a participant-observer in both projects. This helped us pinpoint which technical and organizational areas, such as deployment bottlenecks, handoff practices, data dependencies, and team communication, were most likely to reveal meaningful adoption practices and recurrent barriers. We kept the guide relevant to

the specific challenges of these environments by grounding the study in the daily realities of the teams in both projects.

- **(iii)** We drafted the preliminary guide by organizing open-ended questions into specific thematic blocks. We ensured alignment with the two research questions by covering perceived influential practices, production barriers, and interactions among data, modeling, engineering/MLOps, and organizational coordination. These questions were designed to move from the individual's role to broader organizational and technical practices, allowing participants to describe their daily interactions with tools such as GitLab and Notion in their own words. We focused on creating a non-leading narrative flow that encourages practitioners to reflect on both enablers and friction points they have encountered. The protocol, which forms the foundation of our data collection, is detailed in Table 3.5.

Table 3.5: Interview protocol used in data collection

ID	Theme	Interview Question
Context and Socio-Technical Roles		
Q1	Role & Intersections	What is your current role, and how does your day-to-day work interface with the data science, engineering, and product aspects of the project?
Q2	Evolution under Constraints	How has your role and the team's workflow evolved as the project moved from initial prototyping to production? How did limited resources (compute, team size, tools) shape this evolution?
ML Workflow and Handoffs		
Q3	The Handoff Experience	Can you describe the "handoff" process when moving a model from the experimentation phase to the engineering/deployment phase? Where does the friction or bottleneck usually occur between teams?
Q4	Process Friction	What was the most significant barrier you faced when transitioning the ML model from an experimental notebook/script to a sustained production environment?
MLOps Success Factors and Practices		
Q5	Defining Success	Beyond offline model accuracy, what are the primary indicators you look for to consider a model successfully "operationalized" in production?
Q6	Pragmatic Scaling	What has been the most effective technical or organizational strategy you've adopted to scale and maintain ML components given your project's specific resource constraints?
Coordination, Tooling, and Resilience		
Q7	Artifact Synergy	Can you describe a specific instance where an artifact (e.g., a Notion document, a GitLab issue, or a data contract) acted as an enabler to solve a cross-team communication breakdown?
Q8	Incident Recovery	When a technical failure or silent degradation (like concept drift) occurs in production, what organizational practices or tools allow the team to detect, recover, and learn from it?
External Factors (Data & Governance)		
Q9	Data Governance Trade-offs	How did strict data governance, bureaucratic, and privacy constraints impact your ability to experiment rapidly? How did the team manage the trade-off between compliance and agility?

We selected participants through purposive sampling to target individuals experienced with the phenomenon under study [63]. We defined the inclusion and exclusion criteria in Tables 3.6 and 3.7, respectively, to select practitioners most involved in the end-to-end Machine Learning lifecycle.

Table 3.6: Inclusion criteria for participant selection.

ID	**Criterion**	**Scope**	**Rationale**
IC1	Active involvement in the project (*Brasil Participativo* or *AdsRay*).	Both Cases	Ensures the participant has contextual and practical knowledge of the specific case.
IC2	Occupying roles such as Data Scientist, MLOps Engineer, Data Engineer, Product Owner, Software Engineer, or Infrastructure Engineer	Both Cases	Allows for data triangulation by capturing both technical and management perspectives.
IC3	Direct participation in at least one cycle of ML model development, deployment, or maintenance.	Both Cases	Ensures firsthand experience with the core phenomenon of ML operationalization.

Table 3.7: Exclusion criteria for participant selection.

ID	**Criterion**	**Scope**	**Rationale**
EC1	Team members with exclusively peripheral roles (e.g., administrative, communication, product management) who have no contact with the ML workflow.	Both Cases	Avoids data that does not contribute to the understanding of socio-technical factors in MLOps.

3.6.3 Participant Observation

Throughout the study, the author took part in both projects as a participant-observer, attending team meetings and project activities and contributing to the technical coordination of Case A. This embedded position gave direct access to how decisions were made, documented, and validated, and it produced the meeting transcriptions that we analyzed alongside the documents and interviews. Observation generated evidence in its own right, including the author's own contributions to meetings, and it contextualized the interviews by grounding them in workflows we had seen first-hand. Because an embedded role also carries the risk of bias, we treated the observational material as one source among several and cross-checked it against the interview and document evidence, as detailed in the trustworthiness section below.

3.6.4 Ethics and Consent

Participation was voluntary and based on informed consent. Before each interview, participants were informed of the study's purpose, the intended use of the data, and their right to withdraw, and they gave verbal consent to being recorded. To protect participants, we pseudonymized all individuals by role (for example, "Speaker D, Data Engineering technical lead") and removed direct identifiers from the quoted material. The raw transcripts contained real names, which were replaced before analysis and reporting. Given that the study involved professionals reflecting on their own work practices and posed no more than minimal risk, it did not require submission to a research ethics committee.

3.7 Data Analysis

We analyzed the interview and document data with thematic analysis, following the six-phase process of Braun and Clarke [60]: (i) familiarization, (ii) generating initial codes, (iii) generating themes, (iv) reviewing themes, (v) defining and naming themes, and (vi) reporting. Our coding was hybrid. For the *challenges*, we coded inductively, letting the categories emerge from the practitioners' accounts. For the *practices*, we began deductively from the MLOps practice catalog of Alves et al. [2] (cited throughout as P-xx) and allowed new, previously undocumented practices to emerge inductively where the catalog did not cover what the teams did. This kept the analysis grounded in an established codebook while remaining open to context-specific adaptations, and it let us compare stated practices from the interviews against recorded evidence in the GitLab and Notion artifacts.

In practice, we imported the collected corpus into ATLAS.ti as two document groups, one per case, and coded each item in two passes, first the challenges of operationalizing ML and then the practices adopted in response. Because Case A yielded substantially more evidence than Case B, we coded the two cases separately rather than pooling them. The coding followed the LLM-assisted approach detailed below: for the challenges we ran a purpose-built skill instructed to code the challenges related to ML operationalization of the ML products, to analyze all the provided evidence, and to report the list of codes found, with an analogous skill for the practices. Every proposed code was then read against its source evidence, verified, and refined by the analyst before being retained, and we repeated the same two-pass procedure in Stage 2 over the interview transcripts. For example, a GitLab issue recording the need to "Define the interface contract of the pipeline stages" was coded as the absence of an explicit data contract between producing and consuming teams (DC3).

We used a large language model (Anthropic, Claude Opus 4.8, accessed on July 10, 2026) to assist the analysis, together with ATLAS.ti, to support familiarization, the generation of candidate codes, and the organization of codes into themes, following the recommendations of Ornelas et al. [64]. In line with that guidance, the model supported but did not substitute the interpretive work. We reviewed and validated every code, theme, and quotation, and the model's suggestions were accepted only when they were traceable to the underlying evidence. This approach requires prompting literacy and continuous human oversight, and it carries known risks that we sought to mitigate, namely bias in the model's suggestions, loss of context, limited reproducibility, and rapid model evolution [64]. We addressed these by keeping the human analyst as the final arbiter of every coding decision, by anchoring codes to specific quotations retained in ATLAS.ti, and by maintaining the audit trail described below so that the path from raw evidence to reported theme remains inspectable. Throughout, the model was connected to the ATLAS.ti Model Context Protocol (MCP) server, so that every code, refinement, and theme it helped produce was written directly into the ATLAS.ti project and kept in step with the analyst's decisions.

Figure 3.8 summarizes this LLM-assisted, human-validated process, from evidence collection through to the final themes.

We then used the model to cluster the first-order codes into candidate themes for both challenges and practices, reviewed each grouping against the evidence, and consolidated them into eight themes (T1–T8), with ATLAS.ti as the workspace for this affinity mapping. The challenge themes were subsequently grouped into the four aggregate dimensions of Amrit and Narayanappa [16] (organizational, technical, operational, and business), with the practices adopted filed under the challenge each addresses, yielding the typology of

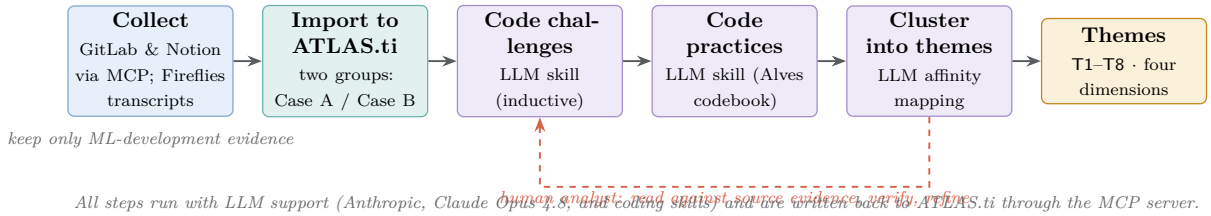


Figure 3.8: The LLM-assisted, human-in-the-loop analysis process

challenge dimensions shown in Figure 3.9.

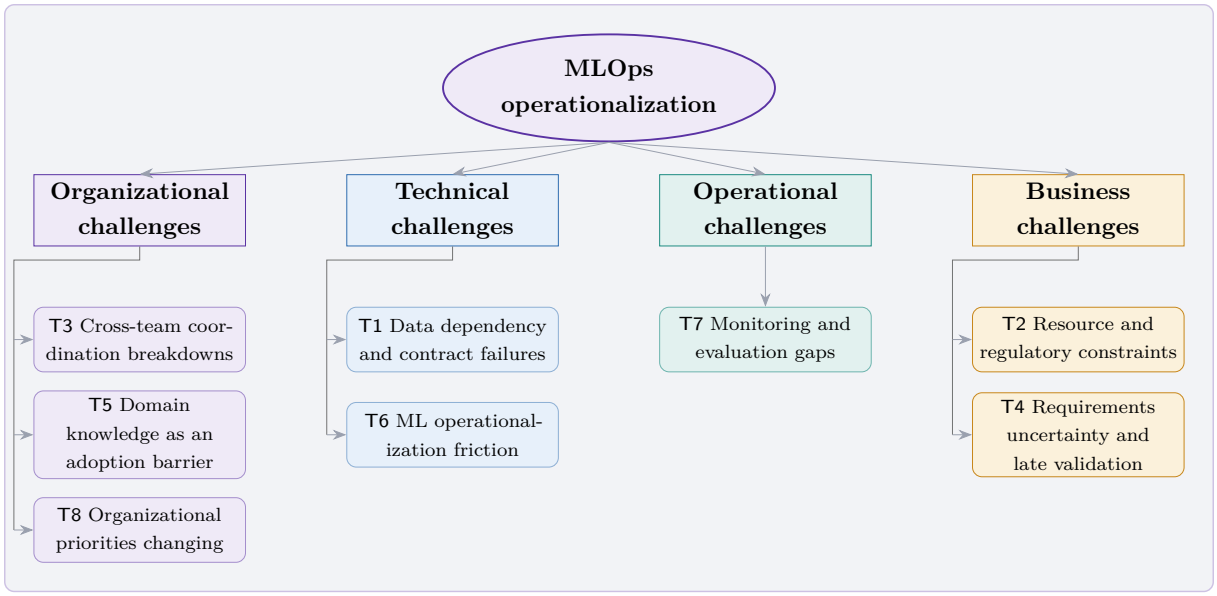


Figure 3.9: Typology of first-time MLOps adoption challenges.

Because the two cases differ in size and in how much of each was observable, we analyzed each case separately and then compared them, reporting convergences and divergences rather than pooling the evidence. Given this asymmetry, cross-case statements are treated as suggestive. The coding was also performed by a single analyst. We discuss the implications of this in the Limitations and describe below the measures taken to support reliability.

Chapter 4

Results

In this chapter, we present the results of our analysis. We conducted a mixed-methods thematic analysis. We collected data from April 2025 to July 2026 across 67 technical artifacts (GitLab and Notion issues, technical design documents) and 16 meeting transcriptions. We conducted and transcribed 9 semi-structured interviews, which allowed us to adjust the order of questions or to draw on relevant findings from one interview to the next. We also observed the teams' workflows to understand the lifecycle of ML development.

We analyzed the data through a mixed thematic analysis method while reflecting on the research questions, using ATLAS.ti, a computer-aided qualitative data analysis software, and employing an LLM-assisted approach to support the analysis, following the guidelines of Ornelas et al. [64] to organize, analyze, and visualize the data. We mapped both the challenges faced and the practices adopted when implementing ML products. For the challenges, we coded the data inductively to allow it to emerge organically from the data. For the practices, we began deductively using the codebook developed by Alves et al. [2], which maps 91 practices related to ML product development, alongside an ad hoc literature review to identify recently documented practices. However, during the coding process, we allowed new and previously undocumented practices to emerge inductively, ensuring our findings were both theoretically grounded and sensitive to practitioners' work.

In Figure 4.1, we illustrate the coding and theme generation process applied to our data, guided by the foundational principles of thematic analysis proposed by Braun and Clarke [60]. Following an initial familiarization with the empirical data (Phase 1), we began the analysis inductively, generating initial codes to capture the real-world challenges reported by practitioners. Subsequently, we applied a deductive approach to code the corresponding practices using the predefined codebook [2]. Moving to Phase 3, we used ATLAS.ti to group both challenges and practices into initial candidate themes. Finally, in Phase 4 (reviewing themes) and Phase 5 (defining and naming themes), we refined these

candidates.

Throughout this work, we refer to participants by the role-based pseudonyms listed in Table 4.1, which reports the nine interviewees.

Table 4.1: Anonymized participants and roles.

ID	**Primary Role**	**Responsibilities**	**Case**
Speaker C	Technical Coordinator	Researcher, responsible for the project roadmap and overall technical alignment.	Case A
Speaker F	Data Science Tech Leader	Lead for model development, MLOps strategy implementation, and pipeline automation.	Case A
Speaker G	Data Science Tech Leader	Lead for requirements gathering and management of stakeholder-facing ML features.	Case A
Speaker R	Data Science Engineer	Responsible for model architecture, training, and deployment processes.	Case A
Speaker D	Data Engineering Tech Leader	Responsible for data pipeline maintenance, database integration, and ingestion logic.	Case A
Speaker E	Data Engineering Tech Leader	Responsible for infrastructure liaison, inter-team coordination, and database access provisioning.	Case A
Speaker S	Subject Matter Expert	Provided domain-specific labeling and Political Science expertise to guide the models.	Case A
Speaker M	ML Engineer	Responsible for the retrieval-augmented classification pipeline, local model deployment, and observability.	Case B
Speaker J	Subject Matter Expert	Research and methodology coordinator, owns the disinformation taxonomy and labeling criteria.	Case B
Speaker P	ML/Data Engineer	Responsible for developing ML products and migrating the classification pipeline from prototype to production.	Case B

Finally, the thematic analysis of the coded excerpts yielded eight first-order code families, which were consolidated into eight themes (T1–T8) reported below, with the practices the teams adopted filed under the challenge each addresses. In total, we coded 302 quotations (226 in Case A and 76 in Case B), with each first-order code organized as a sub-code of its theme.

Figure 4.1 summarizes the analytic structure, mapping the eight themes (T1–T8) to their first-order codes, with each theme's challenge codes shown alongside the practice codes adopted against it. Figure 4.2 reports how frequently each theme was coded in each case (a quotation may be counted under more than one theme and may carry more than one code).

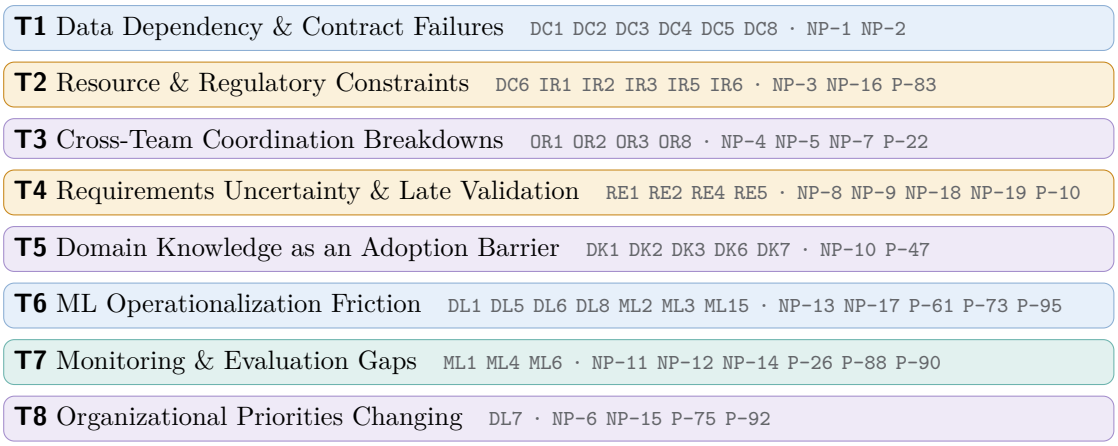


Figure 4.1: Thematic coding structure

The analysis indicated that Case A had more documented evidence than Case B. Therefore, we reported the evidence for each theme separately for each case. Figure 4.2 reports the grounding of each theme per case, where a quotation may be counted under more than one theme and may carry more than one code. Case A has longer participant-observed engagement and prevails across most themes, with operationalization (T6), coordination (T3), requirements uncertainty (T4), and data dependency (T1) most grounded there. Case B contributes a smaller but thematically broad set of quotations. The single theme in which the two cases are comparably grounded is monitoring and evaluation (T7), which was sparsely coded in Case A yet is Case B's densest theme. This asymmetry necessitated reporting each topic separately in the sections below.

	T9... 41	T8:... 7	T5:... 22	T3:... 38	T4:... 32	T6:... 45	T7:... 13	T2:... 30	T1:... 44	Totals
Case A 17 134	39	7	17	35	28	40	6	29	42	243
Case B 2 22	2		5	3	4	5	7	1	2	29
Totals	41	7	22	38	32	45	13	30	44	272

Figure 4.2: Grounding per theme, by case.

Table 4.2: Code frequency by aggregate dimension and theme.

Dimension	**Theme**	**Codes**	**Frequency**
Organizational	T3 Cross-team coordination breakdowns	8	84
	T5 Domain knowledge as an adoption barrier	7	45
	T8 Organizational priorities changing	5	7
	Subtotal	*20*	*136*
Technical	T1 Data dependency & contract failures	8	64
	T6 ML operationalization friction	12	86
	Subtotal	*20*	*150*
Operational	T7 Monitoring & evaluation gaps	9	41
Business	T2 Resource & regulatory constraints	9	67
	T4 Requirements uncertainty & late validation	9	73
	Subtotal	*18*	*140*

Our practice coding began deductively from the MLOps practice codebook of Alves et al. [2], which maps 91 practices across the ML lifecycle. In the constrained, first-time contexts studied here, this base proved incomplete. We mapped nineteen practices that recurred in the evidence but were not documented in the codebook, and we added them inductively as a new family (NP-1–NP-19). The remaining coded practices instantiate existing Alves practices (cited as P-xx). Table 4.3 summarizes these additions by theme: the full first-order practice codebook, with each code's Alves cross-reference, is given in Table A.2 of the codebook appendix.

Table 4.3: Practices added inductively to the Alves et al. codebook, by theme.

Theme	Added practices (NP)	Count
T1 Data dependency & contracts	NP-1, NP-2	2
T2 Resource & regulatory constraints	NP-3, NP-16	2
T3 Cross-team coordination	NP-4, NP-5, NP-7	3
T4 Requirements uncertainty	NP-8, NP-9, NP-18, NP-19	4
T5 Domain knowledge	NP-10	1
T6 ML operationalization friction	NP-13, NP-17	2
T7 Monitoring & evaluation gaps	NP-11, NP-12, NP-14	3
T8 Organizational priorities changing	NP-6, NP-15	2
Total		**19**

In the following sections, we describe the challenges documented in the evidence, how the teams responded, and the practices adopted to address each.

4.1 Organizational challenges

The organizational challenges comprise three themes: cross-team coordination breakdowns (T3), domain knowledge as an adoption barrier (T5), and shifting organizational priorities (T8). Together they account for 136 coded quotations (Table 4.2). Coordination (T3), with 84, is among the most grounded themes in the study, whereas the priorities theme (T8) rests on the least evidence of any theme. Each subsection reports the theme and the practices the teams adopted in response.

4.1.1 T3: Cross-Team Coordination Breakdowns

Coordination across the ML, Data Engineering, Development, and Infrastructure teams was a recurrent source of friction (family OR). It appeared in 84 quotations and was among the densest challenge themes in the study. Knowledge silos (OR2) concentrated capabilities in individuals, unclear ownership boundaries (OR1) left work between teams undefined, and handoff ambiguity at the deployment boundary (OR3) delayed integration. A participant characterized the deployment boundary as requiring "a very dense testing layer that greatly delays deliveries", because work integrated into the platform then had to be validated with stakeholders and staged in a homologation environment (Data Science team meeting, #R15, Case A).

Much of this friction occurred at the boundaries with the Infrastructure team and Dataprev. Infrastructure support was reported to run through a single individual rather than the team, "we manage that contact with infrastructure, but not with the infrastructure

team. It is with one person" (Speaker D, Case A), and the absence of a defined ticket-intake process left demands centralized and overdue. Participant D argued that a roughly five-minute request for WireGuard access "stayed open in their queue for more than three weeks". A second lead singled out Dataprev as "the most complicated area", with "no direct communication channel" (Speaker E, Case A).

The same coordination pattern occurred on the Data Science side, highlighting the coordination problem between the Data Science and the Infrastructure teams. Moving models to production ran "just [by] word of mouth", with "no process, nothing" (Speaker F, 13 Jul 2026, Case A), and the lab infrastructure the team fell back on was "immature [...] a team without well-defined processes, where the processes depend on a single person" who, when absent, left work "in limbo" (Speaker F, Case A). A remote Data Science engineer added that being co-located "would have helped [him] understand the context and the problem faster", so decisions made in isolation were delayed until the weekly meeting for validation and were sometimes redone (Speaker R, Case A).

In Case B, cross-team coordination took the form of translation work between technical and domain roles. The ML Engineers were described as needing "a learning exchange" to bring "a more theoretical and critical lens into the construction of the code". The subject-matter specialist described the same exchange from the domain side, "changing the wheel with the car moving", as "a lot of exchange and learning, explaining to you how it worked and you understanding it to bring it into your universe", crediting the joint participation of the two organizations as "what made it work" by conducting meetings to "show the platform and its functionalities" (Speaker J, Case B).

The introduction of a dedicated product-owner role also improved coordination in Case A. After this, the team "began a more structured discussion of requirements, priorities and focus" with regular model-validation sessions (Speaker C, Case A). The same coordination was observed in Case B, where the teams maintained close, continuous contact, with deployment transitioning from the ML engineer to an infrastructure engineer (Speaker M, Case B).

The most frequently coded response was the adoption of cross-functional coordination mechanisms (NP-4): incorporating domain and data roles into the Data Science team's routine, together with translation work between technical and domain roles (NP-5) and recurring showcase and alignment meetings (NP-7). One meeting framed the incorporation of a political-science contributor as "essential for the model", since interpreting the text alone was insufficient (Data Science team meeting, 10 Feb 2026, #R07, Case A). Similarly, Case B reintegrated a narrative-analysis specialist scoped to "translating complex discursive phenomena into calibration data for the AI under development", and constituted "a restricted operational group [...] for fast alignments and decision-making on the

methodological flows" (AdsRay alignment relatoria, 06 Jul 2026, Case B). These correspond to restructuring teams to be cross-functional (P-97), pairing data science and software engineers (P-98), translation work across roles (P-99), and organizing showcase meetings (P-103). Case B also externalized knowledge through architecture documentation (P-22): the original methodology could be picked up after its authors left "because it had been very well registered", since "that registration shows a project's maturity and the continuity it will have or not" (Speaker J, Case B, P-22, OR2), corresponding to documentation (P-23) in the codebook.

4.1.2 T5: Domain Knowledge as an Adoption Barrier

Building ML features for civic and political content required domain knowledge that the Data Science team lacked. The theme carries 45 coded quotations. The construction of a controlled vocabulary for proposal classification depended on subject-matter experts, and the documentation recorded this dependency directly:

> "We had some difficulties generating the controlled vocabulary that would be used as input for the classification. To generate it, we needed the work of specialists to classify a set of proposals to use as input for the vocabulary."

project documentation (Notion), Case A

In Case B, domain knowledge (DK3) was the main concern. The team maintained a knowledge base built on its five thematic axes (gender and sexuality, ethnic-racial relations, collective health, democratic stability, and the socio-environmental agenda), spanning finer topics such as women, traditional peoples and communities, the Black population, and the LGBTQIAPN+ community, as the ground truth for classification. The recorded difficulty was reconciling human and machine construals of that taxonomy:

> "While for the research team the thematic axes are treated separately and discretely, at the algorithmic level (semantic match), the AI reads the data in an integrated way."

AdsRay alignment relatoria, Case B

This framed the work of understanding how the AI reads and evaluates the content to align what is written conceptually with what the machine processes. The Case B engineer confirmed that the journalism team owned the criteria outright: "they dictated how classification was done", sending an initial spreadsheet of criteria and later updates that the engineers "had to adapt to" (Speaker M, Case B, DK2, NP-10).

The recorded responses were to treat process-defined categories as ground truth (NP-10) and to distribute labeling across multiple personas and specialists (P-47), corresponding to working closely with stakeholders from day one (P-16) and data labeling (P-53).

4.1.3 T8: Organizational Priorities Changing (Delivery Disruption)

Delivery cadence was repeatedly disrupted (family DL). It rests on the least evidence of any theme, seven quotations under a single code. The densest single code in this family was sprint delivery blocked by an inter-team dependency (DL1). This challenge was described as the Data Science team depending on the Data Engineering team to "take the data ... create Gold tables with this structured, clean data, with the columns well related" (Data Science team meeting, #R13, Case A). Integration between the ML deliverables and the main platform was a second, linked difficulty (DL6), with the documentation recording a delivery held in suspension.

> "[The team] keeps the integration of the hate-speech detection model paused, still under evaluation as to the best way to include it in the platform and in the data pipeline."
>
> *project documentation (Notion), Case A*

The interviews recast this disruption as a prioritization problem specific to *self-generated* work. A Case A product researcher observed that ML "was always much more about wanting to innovate than a demand from our partner", carried "no delivery deadline", and so reached production only partially, "not because it was not of good quality, but because it was not a screaming enough pain" (Speaker C, Case A, DL7, DL1). Stakeholders-driven priorities "fell onto features addressing more critical pains", leaving little room to prioritize the platform adaptations ML required (Speaker C, Case A, DL1, DL5). The interviews with the Data Science team corroborated this dynamic: because ML versions take "six, seven months" while the rotating, "immediatist" stakeholder "has more voice", stakeholders' priorities "enter the pipeline first" (Speaker F, Case A, DL1, DL5), and a Data Science engineer recorded that a finished classification model "lost [...] priority" and remained "in homologation" (Speaker R, Case A, DL1, RE2).

The recorded responses were the adoption of a decoupled delivery architecture (P-92), isolating ML deliverables from the platform's release cycle through external orchestrators, and adapting the agile process to the uncertainty of ML work (P-75), including time-boxed tasks (NP-15) and a monthly report as a delivery-accountability mechanism (NP-6).

4.2 Technical challenges

The technical challenges cover two themes: data dependency and contract failures (T1) and ML operationalization friction (T6). With 150 coded quotations between them (Table 4.2), this is the highest-frequency dimension, and operationalization (T6) is the single most

grounded theme in the study. Each subsection reports the challenge and the practices adopted in response.

4.2.1 T1: Data Dependency and Contract Failures

Operationalizing the ML components depended on the data produced and maintained by a separate Data Engineering team, and this dependency was a frequently coded challenge in the study (family DC), grounded in 64 quotations. The evidence recorded three recurrent forms of difficulty in Case A: interruptions in data availability that blocked ML work (DC2), the absence of an explicit data contract between producing and consuming teams (DC3), and changes to data or database schema propagated without prior notification (DC1).

Availability interruptions were a dense sub-theme. The project documentation recorded access blocks and data that was required but unavailable:

> "Some data required to develop the User Categorization model are not available in the Brasil Participativo replica database."

project documentation (Notion), Case A

These interruptions coincided with the absence of an explicit interface between data producers and consumers (DC3). A GitLab issue recorded the need to "Define the interface contract of the pipeline stages". In a meeting, a participant described how database schema changes propagated into downstream breakage, observing that the consuming code "was very hardcoded" and that "every time you make this schema change, all the other things will break" (Data Science team meeting, #R15, Case A).

A Data Engineering interview traced part of these interruptions to how data access was scoped internally. Because permissions were granted per user, a newly created table was "only that user's [...] it is not available to everyone", so a colleague validating the work "would have to download the whole data project and run that pipeline" to obtain the table, which "delays some deliveries and validations" (Speaker D, 10 Oct 2025, Case A). The same interviewee added that the team's local database, rebuilt daily from the Dataprev replica, "sometimes ends up different from production", while restricted replica access left "part of the team" unable to run the analyses it needed (DC2, DC4).

A product researcher interviewed later reinforced the availability problem from a modeling standpoint: with only "the data we have on the platform itself" and "no national database of participatory processes" to train on, the models stayed "very restricted to our own experience" (Speaker C, Case A, DC2, DC4).

The Case A Data Science team interviews recorded the consuming side of this dependency. A Data Science engineer reported that "there was no data pipeline", making the

understanding and versioning of datasets the hardest part of the work, so that he "ended up solving everything manually" by querying the database directly (Speaker R, 13 Jul 2026, Case A, DC3, DC2). A Data Science engineer traced the maturation of data provisioning from "a CSV of proposals" with weak structure to structured JSON collections (proposals, events, organizations) governed by a defined data workflow (Speaker G, Case A, NP-1, DC5).

In response, the Case A teams introduced *Gold* tables as an inter-team data contract (NP-1):

> "We have already created much of the Gold tables required by the RAG team. We are now working on the DAGs to generate and send the JSONs."
>
> *project documentation (Notion), Case A*

In Case B, by contrast, the team owned its data supply and engineered it from the start. The rag-airflow repository recorded 46 open work items each implementing a Scrapy spider for a distinct regional news portal. Every spider issue described the extraction criteria "collect the maximum metadata possible from each article, obligatorily including the author's name" (Case B GitLab work items #148–#217). This automated, high-throughput ingestion (NP-2) contrasts with Case A's recurrent data-availability blocks. These practices correspond to making data available in a central platform as a contract between producer and consumer (P-101) and to defining the data pipeline (P-46) in the Alves et al. codebook.

4.2.2 T6: ML Operationalization Friction

Turning models into running services and keeping them observable was a distinct cluster of difficulty (family ML). Operationalization is the most grounded theme in the study, with 86 coded quotations. Model deployment and API creation (ML3) spanned local model serving, orchestration, and access control. An interview underscored that this deployment work depended on the Infrastructure team: for the classification model the data team "depended on the infra team to grant access, to containerize the solution, and to enable the communication of the API [...] with Brasil Participativo" (Speaker D, 10 Oct 2025, Case A). The same access constraints forced an orchestration workaround. The team could not let the RAG tool read the platform's data directly, so it "had to use Airflow to do that extraction", duplicating a process that would otherwise have been single-step (Speaker D). The recorded responses centered on making the pipeline modular and portable. The teams adopted modular, componentized pipeline design (P-73), which let stages be built and changed independently. A Case A product researcher described building the team's own retrieval-augmented pipeline as the moment experimentation was organized toward

"more critical problems" (Speaker C, Case A, P-73). Where direct platform access was unavailable, the team used external tools (N8N) as an orchestration workaround to bypass the release bottleneck (NP-17), alongside the model training and retraining work itself (ML2). These correspond to modularizing code (P-79) and keeping the infrastructure right (P-85) in the Alves et al. codebook.

4.3 Operational challenges

The operational dimension holds a single theme, monitoring and evaluation gaps (T7), coded in 41 quotations (Table 4.2). It is the least grounded of the substantive challenge themes overall, yet it is the densest theme in Case B, an asymmetry the results below examine.

4.3.1 T7: Monitoring and Evaluation Gaps

Evaluation and monitoring, by contrast, were reported as absent or partial in Case A (ML1, ML4, ML6). A participant stated that objective evaluation had not yet been done but was needed "so that we could evaluate the model in terms of objective metrics ... and also use that data to retrain a new model" (Data Science team meeting, 10 Feb 2026, #R06, Case A). Another recorded that stakeholders "complained that there were no objective criteria", which the team then began to define (Data Science team meeting, 23 Jan 2026, #R03, Case A).

In Case B, however, the team deliberately invested in automated data pipelines and monitoring. The team benchmarked embedding models (NP-11), integrated a retrieval-augmented-generation evaluation harness (RAGAS) and an LLM observability tool (LangFuse), and operated a batch, human-in-the-loop training loop in which:

> "the model starts on a 60-site sample. Training runs in batches — data sent together, tested on the base, manually classified by the research and, after new criteria are inserted, reintroduced into the AI to measure the confidence index."
>
> *AdsRay alignment relatoria, 06 Jul 2026, Case B*

This operational maturity extended to their Continuous Integration pipelines, with dedicated work items to build the Docker image in the pipeline (work item #225), to validate DAG imports before deployment, since "a DAG that fails to import passes the lint stage and only fails when Airflow tries to load it in production" (work item #223), and to enable Secret Detection against the accidental commit of "a real secret (access key, password, token)" (work item #224, Case B). Planned dashboard indicators (updated-news totals,

average disinformative excerpts per volume), a geographic heat map, authorship tracking, and real-time handling of reviewer revisions extended this monitoring orientation.

A Case B ML engineer emphasized the need to move the classification pipeline from an experimentation stack to a production one. According to them, Langflow worked well "for a prototype, for a PoC", but there was "no way to put a classification into production" with it, since it was slow, and its Elasticsearch backend "fell over all the time [...] you could not partition the vector database properly". To migrate to a production RAG, the team had to learn all the tools and techniques, while also re-provisioning the infrastructure (Speaker P, 14 Jul 2026, Case B, DL8, DL6, IR6, P-10). Once migrated, the costly LLM classification "became much faster and more direct" (Speaker P, Case B, NP-2, DL6). This prototype-to-production migration, captured by the inductive code DL8, is the Case B counterpart of Case A's orchestration workaround (NP-17): each is a response to an experimentation stack that would not survive production.

Interviews deepened the evaluation picture on both sides. The Case B ML engineer described the same shift from result-only inspection to per-stage observability, "at first we only looked at the final classification result [and] then began to look at observability of each stage", backed by LLM observability tooling (Speaker M, 10 Jul 2026, Case B, ML4, P-90). The Case B subject-matter specialist described the complementary human side of that monitoring: the main ongoing activity is "evaluating the quality of the AI's analysis", catching "what we call hallucination, whether the AI is getting it right or not" (Speaker J, 14 Jul 2026, Case B, ML4, NP-14, P-61, DK2). In Case B, the work was initially mostly research, and formal metrics were absent at first, because the work "was initially a research". In the initial stages, the team "did not develop definitive result metrics" and validated manually through a human-centered validation with stakeholders, in which the stakeholders received a set of results from the model and validated its responses (Speaker M, Case B, ML1). In this Case, service failures were handled by manual restart to the previous state without data loss and with no automated recovery (Speaker M, Case B, ML15). The same engineer cast evaluation rigor as constitutive of LLM work, emphasizing that, different from traditional software, which is "more traceable, black-and-white", LLM "testing is a mandatory part of the process [and] cannot be missing" (Speaker M, Case B, ML1, P-61, NP-9). In Case A, a product researcher attributed success of the ML product to the real-world impact of the results. The interviewee emphasized the incorporation of contributions into policy and the participation of historically excluded groups as a success indicator.

> "the technical parameters help [...] but to say it made a difference, it has to be impact"
>
> *Speaker C, 10 Jul 2026, Case A, ML1, ML4*

A Case B ML engineer corroborated a two-axis evaluation as a technique necessary to grant quality to the model. The interviewee underlined that LLM is "really a black box [...] and you can't access its thinking". To ensure the results of the model were correct, the team paired technical measurement through LangFuse, "which documents we're returning [...] how much of the LLM we're using", with a human-driven analysis "aligned with the specialists to understand whether the answer makes sense". (Speaker P, 14 Jul 2026, Case B, P-61, P-90, ML4, ML1, DK2).

The three Case A Data Science team interviews recorded that evaluation was practiced, but unevenly by model type. For classifiers, the engineers reported objective evaluation, with manual labeling under an explicit rule, a train/validation/test split with quality "measured [...] using the test set", macro-F1 for an imbalanced classifier, and confusion-matrix calibration to control false positives (Speaker F, Speaker R, Speaker G, 13 Jul 2026, Case A, NP-11, NP-9). The founding Data Science lead reported monitoring the deployed classifier because the training set was not guaranteed to be "representative" of the whole (Speaker F, Case A, ML4, ML6), and a Data Science engineer used F1 to limit the "many false positives" observed on the platform (Speaker G, Case A, NP-11, ML1). For the LLM report-generation work, by contrast, the same team reported no objective metrics.

> "we don't have objective metrics [...] we read what the model generated and are satisfied or not"
>
> *Speaker F, Case A, ML1*

A Data Science engineer recorded that no stakeholder-defined success metric was ever set to the classification model (Speaker R, Case A, ML1, ML4). Overall, the recorded responses were the definition of objective evaluation criteria and the adoption of experiment tracking with MLflow (P-26) in Case A, and comprehensive operational observability in Case B. These correspond to defining metrics and baselines (P-34) and model performance monitoring (P-35 / P-89) in the codebook.

4.4 Business challenges

The business challenges comprise resource and regulatory constraints (T2) and requirements uncertainty and late validation (T4), together 140 coded quotations (Table 4.2). Resource and regulatory constraints (T2), which include data-sovereignty and access limits, are reported first.

4.4.1 T2: Resource and Regulatory Constraints

Operating inside a government context meant that infrastructure and access were externally constrained (family IR). The theme is grounded in 67 quotations. The dominant constraint in Case A was restricted access to services provisioned by the state IT provider, Dataprev (IR2). The project documentation recorded a denied access request:

> "The study indicated the need for higher-level access in the Oracle Infinity tool. In conversation with Dataprev, this access was denied. We are producing more detailed documentation to pass on to them and reach a solution that is viable for both sides."
>
> *project documentation (Notion), Case A*

Participants described the contractual access to Oracle Infinity as "an imposition in the Dataprev contract" that prevented administrator-level configuration and forced every change to be requested externally (Data Science team meeting, 13 Mar 2026, #R11, Case A). Hardware and shared-compute limits (IR1, IR5) were also recorded, including buffer and memory limits when running local language models and deployment difficulties attributed to internal infrastructure.

Interviews with two Data Engineering technical leads sharpened this constraint (interviews Speaker D and Speaker E, Case A). Speaker E framed the entire difficulty as procedural, "all of our difficulties are related to access and bureaucracy", reporting that requests to Dataprev were "usually delivered two to three months after the request", and that initial access to the replicated database had taken "six to eight months". Reaching the Dataprev domain was itself described as a chain of dependencies, with SSH tunneling through WireGuard, then a database user, then an Airflow user, so that "there are several dependencies to access the data". The same interviews reported that the Oracle Infinity migration degraded extraction to aggregated data only, with "no way to extract the data in raw form, it already comes aggregated", which prevented the team from cross-referencing fields and returning precise figures (Speaker D).

In Case B, an ML engineer described the compute-bound constraint from a limited resources perspective. In this Case, the team initially used a HuggingFlow stack, which "consumed too much resource" in production. This forced migration to the Raystack stack and obligated the team to change the architecture entirely because the local proof-of-concepts "were not viable in production" (Speaker M, Case B, IR1, IR6). On the observability side, LangFuse was not working correctly because logs from a ClickHouse metadata store "were filling the disk" and had to be tuned so that the observability tool could run (Speaker M, Case B). A Case A product researcher framed the equivalent constraints as jurisdictional and legal, in which the dependence on Dataprev infrastructure and Oracle Infinity was "a corporate decision, not ours [that] hampers innovation", while

the data-protection law (LGPD) "bureaucratizes" access, obtained only after "a long and bureaucratic process" through the Conecta Gov service delayed the development of new features and models (Speaker C, Case A, IR2, IR3, DC6). The Data Science lead sharpened the same jurisdictional constraint by stating that although Brasil Participativo is hosted on Dataprev, "there was no way to transfer" the ML components there because it is "a public company that, despite being public, still charges for infrastructure", so the models ran on the laboratory hardware instead (Speaker F, Case A, IR2, DL6).

Two responses were recorded. Driven by data-sovereignty concerns, the Case A team adopted lightweight, Portuguese-trained open-source models run locally (NP-3), described in a meeting as studying "a fairly lightweight LLM, trained for Portuguese" (Data Science team meeting, 10 Feb 2026, #R06, Case A). The team also externalized the access constraints into documentation to negotiate with the provider. Case B reached the same local-model posture from two rationales recorded by two roles. An ML engineer cited that a 120-billion-parameter open model was run locally to "avoid API costs" given available infrastructure (Speaker M, 10 Jul 2026, Case B, NP-3, NP-16). The subject-matter specialist cited the aim was "autonomy, control over our content" and "not depending on big techs that are outside here", with traceability so data "isn't sent to third-party servers or monitored" (Speaker J, Case B, DC6, NP-3). The model choice was itself resource-constrained (NP-16): the Case B team continuously measured "RAM, CPU, time" on local infrastructure to favor a local model over a paid API (Speaker M, Case B, NP-16), and Case A worked in an anonymized, mirrored development environment to keep sensitive data controlled (P-83). These map to managing privacy and security (P-41), creating a controlled environment (P-84), and keeping the infrastructure right (P-85) in the codebook.

4.4.2 T4: Requirements Uncertainty and Late Validation

Requirements for the ML features were frequently undefined or in flux (family RE). Requirements churn was coded in 73 quotations. Undefined or dynamically changing requirements (RE1) surfaced as an explicit trade-off between fixing categories in advance and resolving them at execution time. The project documentation recorded a methodology being reworked once its requirements basis was questioned:

> "The writing-assistant methodology had to be revised because it was being done inductively, rather than grounded in data."

project documentation (Notion), Case A

The Case A Data Science team interviews reinforced this uncertainty. The founding Data Science lead described requirements as "very informal", with no formal process fixing

what would be built, "that never existed [...] a point of vulnerability" (Speaker F, Case A, RE1, OR3), and a Data Science engineer highlighted that "it is hard to build requirements for machine learning [...] it is expected that they change over time", so requirements were "never [...] specified in a document" but emerged from successive results (Speaker R, Case A, RE1, P-10).

In Case B, the team planned to "run practical tests with the new descriptions to understand how the algorithm reacts to the criteria" and to fix the depth of the model's justifications only after those tests. The subject-matter specialist described the requirements as taking the form of a criteria-and-examples artifact for the model. In this Case, an existing keyword-and-question table the team added "the explanation for the AI of what we wanted", a description column and columns giving "a disinformative example, and an informative example" (Speaker J, Case B, NP-19, NP-10). Inserting a new narrative then meant "insert the narrative, test, see whether it improved or not, accept or not", a calibration step that "didn't exist in the manual annotation phase and imposes itself once you build a platform" (Speaker J, Case B, NP-9, NP-13).

The teams' principal response was to use prototyping as a requirements elicitation technique (P-10), running an initial version on real data to elicit and validate expectations with stakeholders, alongside explicit use-case scoping and roadmapping (NP-8) and feature prioritization by ICE scoring (NP-18). In Case B, a specification artifact served as a pre-development contract for the model (NP-19). These correspond to applying prototyping to validate and align objectives (P-10) and translating user needs into data needs (P-12).

4.5 Practices adopted across the challenges

Each challenge theme above has been paired with the practices the teams adopted against it, from inter-team data contracts and local model selection to modular pipeline design, orchestration workarounds, and decoupled delivery. Figure 4.3 consolidates that mapping: for each challenge theme, the practices adopted in response and their cross-reference to the Alves et al. codebook.

Answers to the Research Questions

This section consolidates, from the findings reported above, the study's direct answers to the two research questions, cross-referenced to their first-order codes and themes. The challenges of operationalizing ML for the first time under resource constraints answer RQ1, and the practices the teams adopted in response answer RQ2.

Challenge theme (RQ1)	**Practice adopted (RQ2) · Alves et al. codebook**
T1 Data dependency & contracts	Gold tables as inter-team data contract (NP-1), automated large-scale ingestion (NP-2) · P-101, P-46
T2 Resource & regulatory constraints	Local open-source LLM with retrieval (NP-3), resource-constrained model selection (NP-16), anonymized mirrored dev environment (P-83) · P-41, P-84, P-85
T3 Cross-team coordination	Cross-functional roles, translation work, showcase meetings (NP-4/5/7), architecture documentation as knowledge externalization (P-22) · P-97–99, P-103, P-23
T4 Requirements uncertainty	Prototyping as elicitation (P-10), use-case scoping (NP-8), ICE-score prioritization (NP-18), specification artifact as pre-development contract (NP-19), criteria calibration through tests (NP-9) · P-10, P-12
T5 Domain knowledge	Process-defined ground truth (NP-10), multi-persona labeling (P-47), human–machine construal and medium-specific modeling (DK6/7) · P-16, P-53
T6 ML operationalization friction	Modular/componentized pipeline design (P-73), orchestration workaround via N8N (NP-17), model training/retraining (ML2), experiment tracking (P-26), CI/CD pipeline hardening (P-95), model interpretability (P-61), batch human-in-the-loop training (NP-13) · P-79, P-85
T7 Monitoring & evaluation gaps	RAG evaluation harness (NP-12) and LLM observability (P-90), real-time reviewer handling (NP-14), contextual drift monitoring (P-88) · P-34, P-35/P-89
T8 Delivery disruption	Decoupled delivery architecture (P-92) and agile adapted for ML (P-75), with time-boxing (NP-15) and a monthly delivery-accountability report (NP-6)

Figure 4.3: Challenge themes and the practices adopted in response.

RQ1: Answer. In the resource-constrained, first-time adoption contexts of Case A and Case B, the challenges to operationalizing ML were data dependency and contract failures (T1), resource and regulatory constraints (T2), cross-team coordination breakdowns (T3), requirements uncertainty and late validation (T4), domain knowledge as an adoption barrier (T5), ML operationalization and monitoring gaps (T6/T7), and disrupted delivery cadence (T8), which in Case A took the distinctive form of self-generated ML work carrying no external delivery demand or deadline (DL7). These themes recurred across both cases. Monitoring and evaluation (T7) was the lowest of Case A's substantive challenge themes (the least-grounded priorities theme T8 aside) yet the most grounded in Case B, where the difficulty was recast as human–machine construal alignment (DK6), medium-specific modeling (DK7), and contextual concept drift (P-88).

RQ2: Answer. Under constraint the teams could not adopt the full MLOps catalogue. The minimal set of practices that carried ML work to production comprised modular pipeline design (P-73), resource-constrained model selection (NP-16), orchestration workarounds via external tools (NP-17), specification and Gold-table artifacts as inter-team data contracts (NP-19, NP-1), architecture documentation as knowledge externalization (P-22), cross-functional coordination and translation work (NP-4, NP-5, NP-7), prototyping as requirements elicitation (P-10), and a decoupled delivery architecture (P-92). In Case B these were joined by automated large-scale data ingestion (NP-2), CI/CD pipeline hardening (P-95), a RAG evaluation harness and LLM observability (NP-12, P-90), and batch human-in-the-loop training and measurement (NP-13), with criteria calibrated through practical tests (NP-9).

Chapter 5

Discussion

In Chapter 4, the findings reported describe *what* the two first-time, resource-constrained MLOps adoptions encountered. In this chapter, we discuss *what it means*. We interpret the eight themes, relate them to the MLOps and DevOps literature, synthesize across the two cases, and derive implications for research and for practice. After the analysis, the evidence base emerged as asymmetric, with 302 coded quotations, of which 226 are from Case A and 76 are from Case B. Because of this, we present the cross-case statements as suggestive instead of confirmatory. We organize the discussion around the same four aggregate dimensions as the findings (Organizational, Technical, Operational, and Business), interpreting the challenge themes within each, then draw the threads together in a cross-case synthesis, position the study against prior work, answer the research questions, and close with implications, limitations, and future work.

5.1 Organizational challenges

5.1.1 Coordination as a team-structure effect (T3)

Cross-team coordination breakdowns (§4.1.1, T3) are among the most grounded challenges in our data, and are best understood as a *team-structure* effect. Case A sits at **Pattern A/B** on the team-structure ladder of López-Fernández et al. [31], characterized by separate ML, Data Engineering, Development, and Infrastructure departments collaborating *across* a boundary. In this setting, knowledge silos (OR2), unclear ownership (OR1), and the "very dense testing layer that greatly delays deliveries" at the handoff (OR3) are the expected symptoms of a topology in which no single team owns delivery end to end. Conway's Law [29] supplies the mechanism whereby integrating a system across four separately owned teams reproduces those organizational seams as coordination costs, resulting in what Amrit and Narayanappa [16] call "slow processes", where inter-team

dependency drives the delays, and echoing the collaboration challenges Nahar et al. [46] document across teams building ML-enabled systems.

For López-Fernández et al. [31], infrastructure settings that rely on a single person, with no defined process for handling demands, constitute an acute *autonomy deficit*, in which the data team could not complete its own work because a critical dependency lay outside it. This is exemplified by Case A, in which a five-minute WireGuard request "stayed open in their queue for more than three weeks" (Speaker D). This directly supports Luz and Pinto's [24] contention that *collaboration is the binding constraint* of adoption, and extends their claim by showing that, under resource constraints, the specific *structural form* of collaboration becomes the decisive variable. The response codes the teams improvised (cross-functional coordination (NP-4), translation work (NP-5), showcase meetings (NP-7)) are compensations for an absent structure, instantiating the practices of restructuring teams to be cross-functional (P-97), pairing data science and software engineers (P-98), translation across roles (P-99), and showcase meetings (P-103).

The interviews highlight a resource-constrained aspect of the coordination deficit. A new inductive code, key-person dependency (OR8), refers to processes that depend on a single individual and that are disrupted when that individual is unavailable. This code refers also to handoffs that occur solely through informal communication without established procedures. This is distinct from the knowledge silo (OR2), which in López-Fernández et al.'s [31] terms represents a boundary *between* teams, whereas OR8 indicates a single point of failure *within* a team. In limited resources and first-time collaboration settings, informal coordination methods such as word-of-mouth rely on tacit knowledge, co-location, and a sufficiently small team to facilitate direct communication to solve problems. The Data Science engineer interviewed in a remote context (Speaker R) identified this arrangement as fragile, since remote work reduces the physical proximity necessary for effective informal coordination. Consequently, in such environments, it is important to explicitly identify and address single points of failure, and to introduce redundancy, instead of relying on complex or burdensome processes.

Ownership ambiguity also surfaced in the retrieval-augmented (RAG) features both teams built. It was initially unclear who should own and populate the vector database. The vector store sits between the data team, which produces and cleans the data, and the Data Science team, which defines the embedding model that turns that data into vectors. This is the same unclear responsibility boundary that Nahar et al. [46] document between data-science and engineering roles, here specialized to a RAG pipeline. Case A resolved it with a producer–consumer split of the same shape as the Gold-table contract (NP-1). The data team delivered the treated data, and the Data Science team owned the embedding and vectorization step. Treating the vector store as a contract between the two teams

gave the coordination gap a defined interface.

As Figure 5.1 shows, cross-functional coordination (NP-4) forms the strongest link among the 58 Case A quotations that pair a challenge with a practice code, acting as the primary hub. This code is frequently co-coded with sprint-delivery blocks (DL1, the strongest link), knowledge silos (OR2), unclear role boundaries (OR1), and data-availability interruptions (DC2). The recurrence of a single coordination mechanism across distinct challenges in delivery, ownership, and data provides direct evidence that the underlying friction is organizational and requires changes in team structure to address it. However, other flows are more limited and specific to particular challenges. Resource-constrained model selection (NP-16) addresses hardware and shared-compute limits (IR1, IR5), use-case scoping (NP-8) addresses requirements churn (RE1), and process-defined ground truth (NP-10) and translation work (NP-5) address domain-knowledge barriers (DK3, DK2). These pairings result from co-coding on shared quotations, supporting the challenge-to-practice mapping shown in Figure 4.3. Case B evidence reinforces the cross-case analysis by presenting co-codes challenges with practices (49 quotations), but around evaluation and domain pairs. Observability with monitoring (ML4→P-90), taxonomy with process-defined ground truth (DK3→NP-10), and human-in-the-loop training (ML1→NP-13).

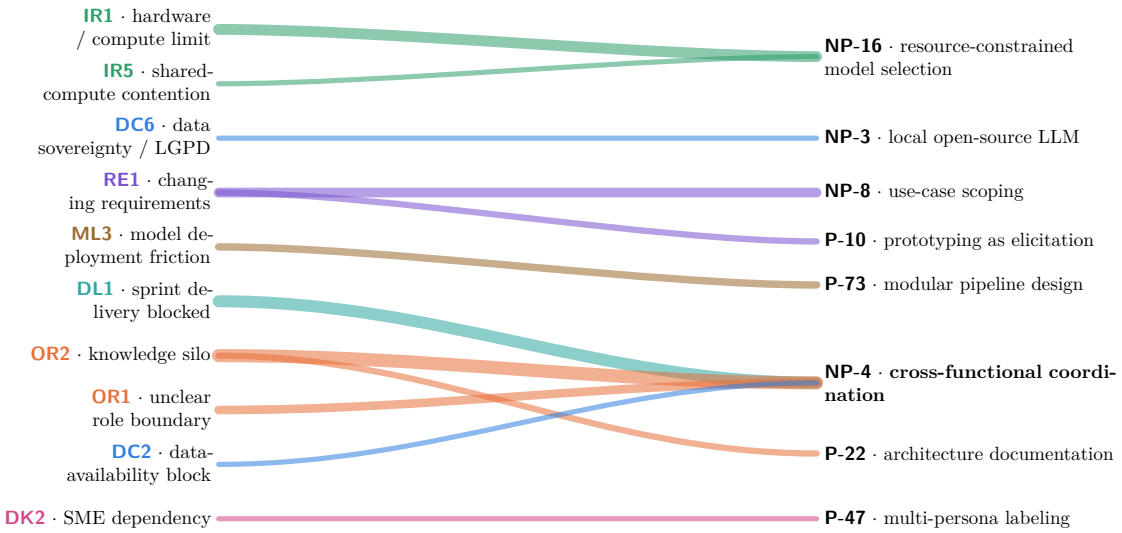


Figure 5.1: Challenge→practice code co-occurrence in Case A

5.1.2 Skill profile and the operationalization gap

A second organizational challenge was the Data Science team's own skill profile. Its technical members were oriented toward data science, strong in modeling and data analysis, with limited experience in the engineering work of taking models to production. Measured against the MLOps role inventory, the team was rich in data-scientist skills and

short on the ML- and DevOps-engineering skills that operationalization demands. The MLOps literature repeatedly names this shortage. It points to the scarce combination of data science, data engineering, software engineering, and operations skills, and to the model-centric mindset that a data-science-heavy team tends to carry [1, 14]. This shortage also underlies the operationalization friction discussed under T6. Much of that difficulty was a matter of engineering practice the team was still acquiring.

The team closed part of this gap through knowledge exchange. Data Science team members learned operationalization from the infrastructure and data engineers, who had more experience running services in production. This upskilling through collaboration belongs to the same family of socio-technical responses as the translation work (NP-5) above, and it supports the reading that, under constraint, the binding factor is how the available people coordinate and share knowledge [24, 46].

5.1.3 Domain knowledge as an adoption barrier (T5)

In the domain-knowledge adoption-barrier theme T5 (§4.1.2), prior work frames domain knowledge as a multidisciplinarity gap [16, 65]. In Case A, this is reflected in the reliance on subject-matter experts to build a controlled vocabulary (DK2). In Case B, however, disinformation detection required reconciling human and machine construals of a thematic taxonomy (DK6), and teaching the model how meaning shifts depending on the specific medium being used (DK7). This suggests that in humanities-grounded ML domains the knowledge barrier is centered in *how* conceptual categories survive translation into a model's representation.

5.1.4 Delivery disruption as a structural symptom (T8)

Delivery disruption (§4.1.3, T8) has usually been attributed to changing organizational priorities and implementation trade-offs [16]. Our evidence supports that, but López-Fernández et al. [31] let us add that because their taxonomy links team topology to software-delivery performance, Case A's disrupted cadence is partly an artifact of its Pattern A/B topology with sprint work blocked on an inter-team dependency (DL1), integration held in suspension at the platform boundary (DL6). The decoupled delivery architecture the team adopted (P-92) by isolating ML deliverables from the platform's release cycle through external orchestrators is, in this light, a way of buying autonomy the team structure did not grant.

The Case A interview adds a second, motivational mechanism. Because ML there was *self-generated*, since the work was innovation-driven and often had no delivery deadline (DL7, §4.1.3), the very autonomy that let the laboratory innovate also removed the external

pull that ordinarily forces work into production, so that deliverables of adequate quality stalled "not because [they were] not good quality, but because [they were] not a screaming enough pain". This nuances Amrit and Narayanappa's [16] "changing priorities" and business-value dimensions. We argue that *demand-less* innovation is structurally starved under resource constraints because it lacks the demand pressure that would justify the cross-team mobilization (infrastructure, product, design) that productionization requires, linking T8 back to the coordination deficit of §5.1.1. It also reframes part of the delivery gap as a *demand-generation* problem, an implication we take up in §5.10.

The Data Science team interviews add a complementary, priority-driven mechanism to this motivational one. Interviewees stated that ML versions take "six, seven months" while a rotating, "immediatist" presidency "has more voice", presidency-favored features "enter the pipeline first" (Speaker F), and a finished feature "lost priority" and remained "in homologation" (Speaker R). Demand-less innovation is starved by a *schedule*, out-competed on a shared backlog by shorter, politically legible deliverables. The turnover of political leadership every few months is itself a source of the "changing priorities" Amrit and Narayanappa [16] treat generically, and is a mechanism specific to the government setting that the constrained Data Science team could neither predict nor control.

This theme also illustrates a caveat that runs through the study's frequency counts. T8 is the least-grounded substantive theme, resting on a single code (DL7) with seven quotations, yet the risk it names, work of adequate quality that never reaches production, is among the most consequential a first-time adopter faces. Frequency records how often an issue was raised, not how much is at stake when it materializes: here the point was surfaced chiefly by one senior participant, the Case A product researcher (Speaker C), and its rarity in the corpus reflects that few others were positioned to see it rather than that it is minor. We therefore read the grounding counts (Figure 4.2, Table 4.2) as measures of salience and recurrence rather than as a ranking of importance. A sparsely coded theme such as T8 can carry more delivery risk than a densely coded one, which is why it is weighted alongside the rest in the recommendations of §5.12.

5.2 Technical challenges

5.2.1 Data dependency and contract failures (T1)

Data dependency and contract failures (§4.2.1 (T1)) are the most densely coded challenges in Case A. Although we group data dependency under the technical dimension, its coded evidence locates the challenge at an organizational-*boundary*. The engineering surface (data contracts, ingestion, gold-table interfaces) is where the difficulty is observed, but its

roots also bridge into the coordination (T3, organizational) and regulatory (T2, business) dimensions. Prior work treats data availability and quality as first-order obstacles in MLOps [16, 59]. Amrit et al. [16] argue that "the entire MLOps output depends on how good your data quality is". While our evidence agrees, we locate the difficulty in previous steps necessary in ML system development. According to our evidence, although the data the Data Science team needed were not low-quality or absent in principle, they were *owned by another party*. In the interviews with the Data Engineering leads (Speaker D, Speaker E), they reported the mechanism used to provide data to the Data Science engineers. The mechanism consists of a newly created table "is only that user's", a colleague must "download the whole data project and run that pipeline" to validate it, and the local base "sometimes ends up different from production". In this context, the codes DC2/DC3 register "availability" and "missing contract" as the absence of a shared interface across an ownership seam. This reflects the inter-team interface mismatch that Lewis et al. [54] describe for ML-enabled systems, and serves as a symptom of the undervalued "data work" identified by Sambasivan et al. [59] as a source of downstream data cascades. The team's response of using Gold tables as an inter-team data contract (NP-1) appears as a coordination artifact between Data Science and Data Engineering teams. This approach instantiates the practice of making data available in a central platform as a producer–consumer contract (P-101) and of defining the data pipeline (P-46) in the Alves et al. practice codebook [2].

5.2.2 Deployment and release friction (T6)

Turning models into services (§4.2.2, T6) reflected the deployment complexity and inter-team dependency challenges identified in prior work [16, 1]. In Case A, ML deployment required support from the infrastructure team to containerize the solution and enable API communication (Speaker D). The same access constraints necessitated an orchestration workaround, using Airflow to feed the RAG. As discussed in §5.1.1, this represents an autonomy deficit at the operational layer, establishing a connection between T6 and T2/T3.

5.3 Operational challenges

5.3.1 Monitoring, evaluation, and the evaluation inversion (T7)

A comparison of evaluation inversion between the cases shows that monitoring and evaluation (T7) is the least prominent substantive challenge theme in Case A, aside from the single-code priorities theme T8, whereas it is the most prominent in Case B (see Figure 4.2). In the three Case A Data Science team interviews (Speaker F, Speaker R and

Speaker G), T7 was coded 18 times. In contrast, the two Case B interviews (Speaker J, Speaker P) yielded 24 codes for T7. Thus, in absolute terms, Case B is more grounded in this theme. When considering the total number of quotations in each case (226 for Case A and 76 for Case B), approximately 8% of Case A quotations address evaluation, compared to 32% in Case B.

More consequentially, the interviews reveal *what* Case A evaluates. Its engineers evaluate for *classifiers* (manual labeling under an explicit rule, train/validation/test splits, macro-F1 on imbalanced data, and confusion-matrix calibration to control false positives (NP-11, NP-9)), but report "no objective metrics" for the *LLM* report-generation work, where validation collapses to reading the output and being "satisfied or not" (ML1, P-61). The binding evaluation gap in Case A appears precisely where outputs are open-ended text, which the earlier product-researcher account had left implicit. (Speaker F, Speaker R, and Speaker G)

This converges with Case B from the opposite direction. The AdsRay team framed testing as *constitutive* of LLM work, "with LLMs testing is part of the process [and] cannot be missing", against traditional software's "more traceable, black-and-white" character, and the team operationalized exactly that intuition with a RAG evaluation harness (NP-12), LLM observability (P-90), and drift framed as an anticipated condition (P-88). A Case B engineer (Speaker P) argued that because the LLM is "a black box" whose reasoning is inaccessible, evaluation is split across two indispensable axes. A technical one is instrumented through LangFuse, and a theoretical one is carried by the journalist specialists (P-61, P-90, DK2). Both cases thus agree that non-deterministic, generative outputs turn evaluation into a first-class development activity, the MLOps-versus-DevOps distinction the literature draws around *non-deterministic testing* and continuous evaluation [1, 16]. What separates them is how far each has *operationalized* that agreement. Case B, owning its pipeline, built tooling around it. Case A, instead, gated by data access and coordination, still relies on human reading for its generative outputs, consistent with the MLOps-maturity view that monitoring capability tracks how far down the pipeline a team has been able to travel [44, 1]. These practices instantiate defining metrics and baselines (P-34) and model performance monitoring (P-35/P-89).

In Case A, the product researcher (Speaker C) also found success in real-world impact. According to him, success "has to be impact", not technical parameters. Even where Case A *does* evaluate rigorously, then, *what counts as a metric at all* remains in the impact of the adoption of the solution instead of purely technical metrics. This framing is specific to the public-value setting of Case A, and it sits outside the pipeline-maturity account.

5.4 Business challenges

5.4.1 Resource and regulatory constraints (T2)

Resource and regulatory constraints (§4.4.1, T2) reinforce this interpretation and, we argue, are the study's clearest departure from standard MLOps narratives. Though prior work notes infrastructure and privacy constraints [16, 1], it treats them largely as technical or compliance concerns. In Case A the constraint is *regulatory* and centered in *data sovereignty*. Because the data live in the state IT provider's (Dataprev) domain, reachable only through a chain of externally granted permissions (SSH tunnelling via WireGuard, a database user, an Airflow user) so that, in the words of Speaker E, "all of our difficulties are related to access and bureaucracy", with requests "delivered two to three months after the request" and initial replica access taking "six to eight months". Data sovereignty is also a recurring challenge that gates the entire pipeline. The team's response of locally run open-source models with retrieval over the platform's own content (NP-3) reads as a sovereignty-driven design choice, instantiating the practices of managing privacy and security (P-41) and keeping the infrastructure right (P-85).

Both cases also ran on academic infrastructure, which was less mature and less provisioned than a large enterprise's. Hardware was scarce relative to production needs (IR1, IR5), and the services the teams ran on it were often unstable. These are everyday constraints of doing first-time MLOps inside a university, and they shaped what the teams could reliably deploy.

5.4.2 Requirements uncertainty and late validation (T4)

Requirements uncertainty (§4.4.2, T4) is well documented in the literature for machine learning systems [8, 47, 49, 50, 51], with provisional objectives and late validation. Given this, the real contribution lies in how the teams handled the situation. In Case A, the team used prototyping as a requirements-elicitation technique (P-10), and, in Case B, criteria calibrated through practical tests (NP-9). Aligning with the expectation-management difficulties attributed to ML's opacity by Amrit and Narayanappa [16] and Silva et al. [66], both cases elicited requirements through iterative validation with stakeholders, applying prototyping to align objectives (P-10) and translating user needs into data needs (P-12).

5.5 Practices adopted as the study's RQ2 contribution

The practices adopted across the challenge themes (§4.5) are where the findings become prescriptive material. Read as a set, the practices form an emergent repertoire for

keeping ML work moving under constraint: modular pipeline design (P-73), resource-constrained model selection (NP-16), orchestration workarounds (NP-17), specification and Gold-table artifacts as contracts (NP-19, NP-1), and architecture documentation as knowledge externalization (P-22). This mirrors, at the level of MLOps, Luz and Pinto's [24] enabler/outcome structure for DevOps where the practices are enablers (sharing, transparency, a lightweight automation posture) whose intended outcomes are the agility and resilience their cases lacked. Because these practices were adopted *reactively*, in response to specific boundary and access failures, they are strong candidates for a proactive "practices for constrained, first-time MLOps adoption" model, a direction we return to under future work.

5.6 Cross-case synthesis

Read together, the two cases converge on their challenge structure and diverge in instructive ways. *Convergence:* coordination (T3), requirements uncertainty (T4), domain knowledge (T5), deployment and release friction (T6), and resource/regulatory constraints (T2) recur across both settings, which, given how different a government participation platform and a disinformation-detection RAG are, suggests these themes capture something general about first-time, constrained MLOps. The points of divergence are the more instructive part of the comparison. In Case A, the **evaluation inversion** (§5.3.1) concentrates its evaluation effort on classifiers while leaving its generative (LLM) outputs largely unmeasured. This contrasts with Case B, where LLM output evaluation is a constitutive part of the delivery. T7 proportionally accounts for roughly 8% of Case A's quotations against 32% of Case B's. In Case A, **data-posture inversion** is the dominant difficulty, since the team was usually *blocked* on a data dependency (waiting on another team and on external data providers), whereas Case B *owned* its ingestion, engineering extraction from 46 news portals (NP-2). Case A ML was self-generated laboratory innovation with a low external delivery demand (DL7), whereas Case B's was a contracted research deliverable with an explicit scaling target. We read these divergences as consequences of *position in the pipeline, control over the data boundary, and the presence of external demand.* They do not reflect differences in team competence. The case that controlled its data boundary and answered a concrete demand (B) advanced to evaluation concerns, while the case that did neither (A) remained bound by access, coordination, and its own discretionary agenda.

Placed on the Google Cloud MLOps maturity model [44], both cases sit low on the scale, which makes this divergence concrete. None of the four ML components reached Level 2, the CI/CD-integrated pipeline. Case A's content-moderation and classification models operate around Level 0, where training and deployment are manual and retraining

is a batch effort triggered by hand, and its report generator, though containerized and orchestrated, likewise has no automated retraining loop. Case B's classification pipeline is the most automated of the four, near Level 1, with Airflow scheduling extraction and enrichment on a fixed cadence, though it too lacks the CI/CD and drift monitoring that mark Level 2. One caveat applies. The Google model is built around *continuous training*, and three of the four components are retrieval-augmented or agentic systems with no classical training loop, so we read them on pipeline automation and delivery rather than on retraining. Seen this way, the maturity gap and the evaluation inversion (§5.3.1) are the same fact viewed twice. Case B travelled further down the pipeline and reached the monitoring and evaluation concerns that Level 2 formalizes, whereas Case A, gated at the data boundary, did not.

These divergences also have a team-structure reading. Case B was a small, co-located team with few silos, closer to the autonomous, cross-functional end of the topology ladder (Pattern C/D) than Case A's interdepartmental Pattern A/B. Members communicated directly, so data and infrastructure demands rarely stalled, and the requirements owners, the subject-matter specialists, worked alongside the ML engineers and supplied a competent, methodologically grounded source of demand. Case A's requirements churned under self-generated, innovation-driven goals (DL7). In Case B, the people who defined the requirements sat next to the people who built against them, and elicitation went more smoothly. Case B's more autonomous team structure came with much less of the coordination and requirements friction that dominated Case A.

This cross-case claim is offered as an interpretation consistent with the coded evidence, tempered by Case B's small corpus (n≈76). It points to a pattern without confirming one.

5.7 Positioning against prior work and contribution

Much of what the two cases surfaced confirms the established MLOps/DevOps account. Data quality and availability, deployment complexity, inter-team dependency, and collaboration difficulties are all recognized [16, 1, 35, 46], and the primacy of collaboration over tooling restates Luz and Pinto [24]. The study's contribution is the *contextual wedge*, examining first-time adopters in a *resource-constrained* setting, and from that vantage two things that the mainstream, enterprise-centered literature under-weights move to the foreground. (i) **Regulatory access as a first-order challenge** (T2): where prior work treats infrastructure and compliance as background conditions, here the jurisdictional data boundary (Dataprev) gates the entire pipeline and shapes model choice. (ii) **Coordination as a team-structure problem with a named remedy**: by joining Luz and Pinto's collaboration-as-core with López-Fernández et al.'s [31] topology ladder, the

study locates a ML organization at Pattern A/B and identifies the horizontal Platform team as the concrete structural correction, a bridge between the DevOps team-topology literature [30] and MLOps adoption that, to our knowledge, has not been drawn for a resource-constrained first-time adopter. The repertoire of practices adopted is the study's prescriptive contribution toward answering *how* such teams cope.

5.8 Answers to the research questions

RQ1 (challenges). In first-time, resource-constrained MLOps adoption, the binding challenges were the *organizational and jurisdictional boundaries* around it: data dependency and contract failures (T1), regulatory/resource access (T2), and cross-team coordination (T3) form a mutually reinforcing core, with requirements uncertainty (T4), domain knowledge (T5), deployment and release friction and monitoring gaps (T6/T7), and delivery disruption (T8) following from, and interacting with, that core. The picture that emerges is of ML teams whose progress is gated by their lack of *autonomy* over data, infrastructure, and delivery.

RQ2 (practices). The practices that enabled deployment were, correspondingly, a minimal set of mechanisms for *manufacturing autonomy under constraint*: inter-team data contracts (NP-1, NP-19), decoupled delivery (P-92) and orchestration workarounds (NP-17) that route around dependencies, cross-functional and translation practices (NP-4/NP-5/NP-7) that compensate for a non-autonomous topology, and in Case B owning the data pipeline and its evaluation loop outright. Interpreted through López-Fernández et al. [31], these are workarounds for the autonomy that a Pattern C/D topology would have supplied by design.

5.9 Implications for research

The study extends the MLOps-challenge literature in a setting of first-time, resource-constrained organizations and shows that a challenge the enterprise literature backgrounds (jurisdictional data access) can be the dominant one. Methodologically, it demonstrates the value of *joining two adjacent literatures*: the MLOps-challenge typologies [16, 1] describe *what* goes wrong, while the DevOps team-topology work [31, 24] explains *why* (structure) and *what to do* (topology change). Bring them together for ML-enabled systems is a direction we commend to future empirical work, ideally with cases that vary team autonomy deliberately.

5.10 Implications for practice

For practitioners adopting MLOps for the first time under constraint, the findings motivate concrete, evidence-anchored recommendations. First, treat **data and delivery boundaries as the primary design problem**, recommending the practices of establish an inter-team data contract early (Gold tables / specification artifacts, P-101, P-46) and decouple ML delivery from the host platform's release cycle (P-92). Second, and most consequential, address coordination *structurally*: the single-person Infrastructure bottleneck (Speaker D) is the anti-pattern a **horizontal Platform team**, providing infrastructure and IT-operations as a self-service, is designed to dissolve [31], moving the ML/data teams toward the autonomy of Pattern C/D. Third, invest in the *socio-technical* practices the teams improvised only reactively (cross-functional roles, translation work, and showcase rites (P-97–99, P-103)) as up-front choices. Fourth, in regulated or sovereignty-sensitive contexts, plan for access latency as a first-class project risk and favor locally controllable model choices (NP-3, P-41, P-85). Fifth, for *self-generated* ML initiatives, deliberate outward communication with showcasing working prototypes to prospective partners, as the Case A researcher proposed, is a lever for moving mature work off the shelf and into use. These recommendations follow directly from the coded findings and are offered as adoption guidance instead of a validated interventions.

5.11 Implications for education

The findings also carry implications for how ML and software-engineering practitioners are trained. The operationalization gap we observed (§5.1.2), in which teams strong in modeling and data analysis were short on the ML- and DevOps-engineering skills that production demands, mirrors the MLOps literature's observation that operations and productionization are typically absent from data-science education [1, 14]. Curricula that pair data-science and modeling training with software engineering, data engineering, and operations (versioning, CI/CD, and monitoring) would narrow this gap directly. Because the binding constraint in both cases was socio-technical rather than algorithmic, ML education should also treat cross-functional collaboration as a first-class competency: data contracts and handoffs between producing and consuming teams (§5.2.1), and the translation work between technical and domain roles that recurred across both cases (§5.1.3). Finally, teaching oriented to resource-constrained, first-time adoption, where a small team must cover the full lifecycle and cannot stand up the complete MLOps stack, would prepare practitioners for the pragmatic, minimum-viable practices this study surfaces, not only the enterprise-scale tooling that dominates existing material.

5.12 Recommendations and lessons learned

The coded evidence supports a compact set of recommendations for teams beginning first-time MLOps adoption under resource constraints. Each addresses one of the challenge themes, pairs it with the practice the two teams used to contain it, and traces that practice to the Alves et al. codebook [2] and, where relevant, to prior work. Together they turn the challenge-to-practice mapping of Figure 4.3 into actionable, cited guidance. Table 5.1 summarizes them. They are offered as risk-mitigation guidance grounded in the two cases.

Table 5.1: Recommendations for constrained, first-time MLOps adoption, by challenge, with grounding.

Challenge	**Recommendation**	**Grounding**
T1 Data dependency & contract failures	Establish an inter-team data contract early (Gold tables or specification artifacts) so consuming teams are not blocked on data they do not own.	NP-1 · P-101, P-46 · [54, 59]
T2 Resource & regulatory constraints	Treat access latency and data sovereignty as first-class project risks. Favor lightweight, locally controllable open-source models with retrieval.	NP-3 · P-41, P-85
T3 Cross-team coordination breakdowns	Add cross-functional roles, translation work, and recurring showcase and alignment meetings. Where feasible, introduce a horizontal Platform team to restore autonomy.	NP-4/NP-5/NP-7 · P-97–99, P-103 · [31, 24]
T4 Requirements uncertainty & late validation	Elicit and validate requirements through prototyping on real data, scope use cases explicitly, and calibrate criteria through practical tests.	P-10, NP-8, NP-9 · P-10, P-12
T5 Domain knowledge as an adoption barrier	Treat process-defined categories as ground truth and embed subject-matter experts in the team's routine, distributing labeling across specialists.	NP-10, P-47 · P-16, P-53
T6 ML operationalization friction	Modularize the pipeline, adopt experiment tracking, and harden the CI/CD path. Use batch human-in-the-loop training where full automation is out of reach.	P-73, P-26, P-95, NP-13 · P-79, P-85
T7 Monitoring & evaluation gaps	Make evaluation a first-class activity, especially for generative outputs: add a RAG-evaluation harness and LLM observability, and watch for contextual drift.	NP-12, P-90, P-88 · P-34, P-35/P-89
T8 Delivery disruption	Decouple ML delivery from the host platform's release cycle through external orchestrators, and adapt the agile process with time-boxing.	P-92, P-75, NP-15

Across the themes, the recurring lesson is that the binding difficulty of first-time adoption under constraint was *organizational and jurisdictional*. If we were to begin either project again, we would treat the ownership of data and delivery boundaries and the structure of the teams as first-order design decisions, on the same footing as the model itself: establish inter-team data contracts before building the first pipeline, negotiate

infrastructure and data access as an explicit, scheduled project risk, and stand up a lightweight evaluation loop for generative outputs from the start instead of deferring it until the outputs are already in use.

Chapter 6

Conclusion

In this work, we examine, from the inside, how small teams adopt MLOps for the first time under resource constraints. Through an exploratory qualitative multi-case study of two contrasting settings, Case A, a Brazilian federal-government civic-tech participation platform (Brasil Participativo), and Case B, a disinformation-detection system (AdsRay), we inductively coded 302 quotations from meeting transcripts, project documentation, task issues, and interviews. We consolidated them into eight themes (T1–T8) spanning the challenges of first-time adoption (RQ1) and, filed under each challenge, the practices that enabled deployment (RQ2).

The central conclusion is that, in these settings, the binding difficulty of MLOps was *organizational* and *jurisdictional*. The intrinsic complexity of machine learning was seldom the limiting factor. Data dependency and contract failures (T1), regulatory and resource access (T2), and cross-team coordination (T3) formed a mutually reinforcing core from which the remaining challenges, requirements uncertainty (T4), domain knowledge (T5), operationalization and monitoring gaps (T6/T7), and delivery disruption (T8), largely followed. Read together, these describe teams whose progress was gated by their lack of *autonomy* over data, infrastructure, and delivery. The models themselves were seldom the constraint. In answer to RQ2, the practices that moved ML work toward production were correspondingly a minimal set of mechanisms for manufacturing autonomy under constraint: inter-team data contracts, decoupled delivery and orchestration workarounds, and cross-functional and translation practices that compensated for a non-autonomous team structure.

The study makes three contributions. First, it brings empirical evidence on MLOps adoption from a setting the literature rarely reaches (first-time, resource-constrained) and shows that a challenge the enterprise-centered literature backgrounds, *jurisdictional data access*, can be the dominant one. Second, it reframes cross-team coordination as a *team-structure* problem with a named remedy, joining Luz and Pinto's [24] collaboration-

as-core account with the team-topology ladder of López-Fernández et al. [31] to locate a civic-tech ML organization at an interdepartmental pattern and to identify the horizontal Platform team as the structural correction that restores autonomy. Third, it distills an emergent repertoire of practices adopted for constrained, first-time MLOps adoption, cross-referenced to the Alves et al. codebook [2], as practical guidance for teams beginning the same journey.

Beyond the specific findings, the study suggests a broader reorientation for first-time MLOps adopters under constraint. They should treat team organization and ownership of data and delivery boundaries as a first-order design concern, on the same footing as the technical build.

6.1 Threats to Validity, Reliability, and Limitations

Criteria for judging the quality of a qualitative research design bear on its *validity*, the accuracy of the findings, and its *reliability*, the consistency of the procedures and the analyst's approach. We assess this study against the quality criteria for qualitative research of Lincoln and Guba [67], complemented by the case-study guidelines of Runeson and Höst [61].

Credibility, the extent to which the conclusions are supported by rich, multivocal evidence, was pursued through data triangulation across three source types, project documents and GitLab/Notion issues, meeting transcriptions, and semi-structured interviews, in both cases, with every theme anchored to traceable quotations retained in ATLAS.ti.

Resonance, the extent to which the conclusions make sense to the studied community, was addressed by using the semi-structured interviews as a validation stage. The challenges and practices that emerged from the documents and meetings were put to practitioners, whose accounts corroborated, qualified, or extended them.

Transferability, whether the findings could plausibly apply elsewhere, rests on analytical generalization. The two contrasting cases, a government civic-tech platform and a disinformation-detection RAG, build a picture of resource-constrained, first-time adoption that is substantive to the studied contexts and may transfer to organizations sharing those characteristics.

Usefulness, the extent to which the study offers actionable guidance to researchers, practitioners, and educators, is served by the repertoire of practices adopted, the challenge-to-practice mapping, and the implications drawn for each of those audiences (§5.9, §5.10, and §5.11).

Dependability, the requirement that the process be systematic and traceable, was supported by an audit trail maintained in ATLAS.ti that links each theme back to the raw

evidence, and by following the six-phase thematic-analysis process of Braun and Clarke [60] over the documented codebook of Alves et al. [2].

Reliability, the consistency of the analytic approach, is the study's principal methodological threat. In this work, a single analyst performed the coding without an independent second coder, so we did not establish inter-rater reliability. We mitigated this by anchoring every code to a specific quotation and by keeping the audit trail above.

Limitations. Four limitations further bound the interpretations. First, the evidence base is *asymmetric* (226 coded quotations for Case A against 76 for Case B), so every cross-case claim is suggestive rather than confirmatory, and the divergences we emphasize may partly reflect how much of each case was observable to us rather than differences in the phenomena. Second, the *participant-observer* stance affords depth but risks selective attention and rapport bias, and the same insider conducted the interviews. Third, a *single analyst* performed the coding without an independent second coder, limiting inter-rater reliability, as noted above. Finally, both cases are Brazilian and government-adjacent, so the regulatory-access findings may not transfer to private-sector or differently regulated settings.

6.2 Future Work

Three directions follow. First, **code Case B to parity** and add further constrained, first-time cases, so the cross-case inversions can be tested rather than suggested, and so the challenge structure can be examined for stability across domains. Second, **operationalize the repertoire of practices adopted** into a lightweight, proactive *model* for constrained MLOps adoption, in the theory→model→application spirit of Luz and Pinto [24], and evaluate it in a new adoption, for instance through a focus group. Third, **test the topology intervention**: study whether introducing a horizontal Platform team measurably restores autonomy and improves delivery cadence in an ML-enabled organization, connecting the qualitative findings here to the delivery-performance measures of López-Fernández et al. [31].

References

[1] Kreuzberger, Dominik, Niklas Kühl, and Sebastian Hirschl: *Machine Learning Operations (MLOps): Overview, Definition, and Architecture*, May 2022. `http://arxiv.org/abs/2205.02302`, visited on 2026-04-22, arXiv:2205.02302 [cs]. v, 1, 2, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 60, 62, 63, 64, 66, 67, 68

[2] Alves, Isaque, Leonardo AF Leite, Paulo Meirelles, Fabio Kon, and Carla Silva Rocha Aguiar: *Practices for managing machine learning products: A multivocal literature review.* IEEE Transactions on Engineering Management, 71:7425–7455, 2023. v, 3, 6, 19, 36, 39, 42, 62, 69, 72, 73, 81

[3] Paleyes, Andrei, Raoul Gabriel Urma, and Neil D. Lawrence: *Challenges in Deploying Machine Learning: a Survey of Case Studies.* ACM Computing Surveys, 55(6):1–29, July 2023, ISSN 0360-0300, 1557-7341. `http://arxiv.org/abs/2011.09926`, visited on 2026-05-04, arXiv:2011.09926 [cs]. v, 1

[4] Eken, Beyza, Samodha Pallewatta, Nguyen Tran, Ayse Tosun, and Muhammad Ali Babar: *A multivocal review of mlops practices, challenges and open issues.* ACM Comput. Surv., 58(2), September 2025, ISSN 0360-0300. `https://doi.org/10.1145/3747346`. v, 2, 19, 20

[5] Symeonidis, Georgios, Evangelos Nerantzis, Apostolos Kazakis, and George A. Papakostas: *MLOps – definitions, tools and challenges.* In *Proceedings of the 12th IEEE Annual Computing and Communication Workshop and Conference (CCWC)*, pages 453–460. IEEE, 2022. v, 11, 12, 14, 15, 16

[6] Kumar, Gaurav and Pradeep Kumar Bhatia: *Comparative analysis of software engineering models from traditional to modern methodologies.* In *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, pages 189–196. IEEE, 2014. 1

[7] Ruparelia, Nayan B: *Software development lifecycle models.* ACM SIGSOFT Software Engineering Notes, 35(3):8–13, 2010. 1

[8] Amershi, Saleema, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann: *Software Engineering for Machine Learning: A Case Study.* In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, Montreal, QC, Canada, May 2019. IEEE, ISBN 978-1-7281-1760-7. `https://ieeexplore.ieee.org/document/8804457/`, visited on 2025-09-09. 1, 2, 5, 6, 10, 17, 19, 64

[9] Hesenius, Marc, Nils Schwenzfeier, Ole Meyer, Wilhelm Koop, and Volker Gruhn: *Towards a software engineering process for developing data-driven applications.* In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 35–41. IEEE, 2019. 1, 6

[10] Akkiraju, Rama, Vibha Sinha, Anbang Xu, Jalal Mahmud, Pritam Gundecha, Zhe Liu, Xiaotong Liu, and John Schumacher: *Characterizing machine learning processes: A maturity framework.* In *International conference on business process management*, pages 17–31. Springer, 2020. 1

[11] Steidl, Monika, Michael Felderer, and Rudolf Ramler: *The pipeline for the continuous development of artificial intelligence models—current state of research and practice.* Journal of Systems and Software, 199:111615, 2023. 1, 2, 12, 14, 16

[12] Shankar, Shreya, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran: *"we have no idea how models will behave in production until production": How engineers operationalize machine learning.* Proceedings of the ACM on Human-Computer Interaction, 8(CSCW1):1–34, April 2024, ISSN 2573-0142. `http://dx.doi.org/10.1145/3653697`. 1, 2

[13] Sculley, David, 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. 1, 2, 5, 6, 7, 9, 10, 18

[14] Mäkinen, Sasu, Henrik Skogström, Eero Laaksonen, and Tommi Mikkonen: *Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?* In *Proceedings of the 1st IEEE/ACM Workshop on AI Engineering – Software Engineering for AI (WAIN)*, pages 109–112. IEEE, 2021. 2, 10, 12, 14, 15, 60, 68

[15] Arpteg, Anders, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch: *Software engineering challenges of deep learning.* In *2018 44th euromicro conference on software engineering and advanced applications (SEAA)*, pages 50–59. IEEE, 2018. 2, 6

[16] Amrit, Chintan and Ashwini Kolar Narayanappa: *An analysis of the challenges in the adoption of mlops.* Journal of Innovation Knowledge, 10(1):100637, 2025, ISSN 2444-569X. `https://www.sciencedirect.com/science/article/pii/S2444569X24001768`. 2, 3, 19, 20, 37, 57, 60, 61, 62, 63, 64, 66, 67

[17] Karpathy, Andrej: *Software 2.0.* Medium blog, 2017. `https://karpathy.medium.com/software-2-0-a64152b37c35`, Accessed: 2026-01-29. 5

[18] Bosch, Jan, Helena Holmström Olsson, and Ivica Crnkovic: *Engineering ai systems: A research agenda.* Artificial intelligence paradigms for smart cyber-physical systems, pages 1–19, 2021. 6

[19] Nushi, Besmira, Ece Kamar, Eric Horvitz, and Donald Kossmann: *On human intellect and machine failures: Troubleshooting integrative machine learning systems.* In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 7

[20] Andrist, Sean, Dan Bohus, Ece Kamar, and Eric Horvitz: *What went wrong and why? diagnosing situated interaction failures in the wild.* In *International Conference on Social Robotics*, pages 293–303. Springer, 2017. 7

[21] Díaz, Jessica, Jorge Pérez, Isaque Alves, Fabio Kon, Leonardo Leite, Paulo Meirelles, and Carla Rocha: *Harmonizing devops taxonomies — a grounded theory study.* Journal of Systems and Software, 208:111908, 2024, ISSN 0164-1212. `https://www.sciencedirect.com/science/article/pii/S0164121223003035`. 7, 8, 9, 19

[22] Iden, Jon, Bjornar Tessem, and Tero Päivärinta: *Problems in the interplay of development and IT operations in system development projects: A Delphi study of Norwegian IT experts.* Information and Software Technology, 53(4):394–406, 2011. 7

[23] Jabbari, Ramtin, Nauman bin Ali, Kai Petersen, and Binish Tanveer: *What is DevOps? A systematic mapping study on definitions and practices.* In *Proceedings of the Scientific Workshop Proceedings of XP2016*, pages 1–11. ACM, 2016. 7

[24] Luz, Welder Pinheiro, Gustavo Pinto, and Rodrigo Bonifácio: *Adopting DevOps in the real world: A theory, a model, and a case study.* Journal of Systems and Software, 157:110384, 2019. 8, 58, 60, 65, 66, 67, 69, 71, 73

[25] Leite, Leonardo, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles: *A survey of devops concepts and challenges.* ACM Comput. Surv., 52(6), November 2019, ISSN 0360-0300. `https://doi.org/10.1145/3359981`. 8, 9

[26] Fitzgerald, Brian and Klaas Jan Stol: *Continuous software engineering: A roadmap and agenda.* Journal of Systems and Software, 123:176–189, 2017. 8, 18

[27] Shahin, Mojtaba, Muhammad Ali Babar, and Liming Zhu: *Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices.* IEEE Access, 5:3909–3943, 2017. 8

[28] Macarthy, Ruth W. and Julian M. Bass: *An empirical taxonomy of DevOps in practice.* In *Proceedings of the 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 221–228. IEEE, 2020. 9

[29] Conway, Melvin E: *HOW DO COMMITTEES INVENT?* 1968. 9, 19, 57

[30] Shahin, Mojtaba, Mansooreh Zahedi, Muhammad Ali Babar, and Liming Zhu: *Adopting continuous delivery and deployment: Impacts on team structures, collaboration and responsibilities.* In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 21–30. ACM, 2017. 9, 67

[31] López-Fernández, Daniel, Jessica Díaz, Javier García, Jorge Pérez, and Ángel González-Prieto: *DevOps team structures: Characterization and implications.* IEEE Transactions on Software Engineering, 48(10):3716–3736, 2022. 9, 19, 57, 58, 60, 66, 67, 68, 69, 72, 73

[32] Perera, Pulasthi, Roshali Silva, and Indika Perera: *Improve software quality through practicing DevOps.* In *Proceedings of the 17th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 1–6. IEEE, 2017. 9

[33] Breck, Eric, Shanqing Cai, Eric Nielsen, Michael Salib, and David Sculley: *The ML test score: A rubric for ML production readiness and technical debt reduction.* In *Proceedings of the IEEE International Conference on Big Data*, pages 1123–1132. IEEE, 2017. 10, 16

[34] Polyzotis, Neoklis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich: *Data lifecycle challenges in production machine learning: A survey.* ACM SIGMOD Record, 47(2):17–28, 2018. 10

[35] Paleyes, Andrei, Raoul Gabriel Urma, and Neil D. Lawrence: *Challenges in deploying machine learning: A survey of case studies.* ACM Computing Surveys, 55(6):1–29, 2022. 10, 15, 16, 66

[36] Martínez-Fernández, Silverio, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner: *Software engineering for AI-based systems: A survey.* ACM Transactions on Software Engineering and Methodology, 31(2):1–59, 2022. 10

[37] Lwakatare, Lucy Ellen, Aiswarya Raj, Ivica Crnkovic, Jan Bosch, and Helena Holmström Olsson: *Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions.* Information and Software Technology, 127:106368, 2020. 10, 14, 16

[38] Lwakatare, Lucy Ellen, Ivica Crnkovic, and Jan Bosch: *DevOps for AI – Challenges in development of AI-enabled applications.* In *Proceedings of the International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6. IEEE, 2020. 10

[39] Tamburri, Damian A.: *Sustainable MLOps: Trends and challenges.* In *Proceedings of the 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 17–23. IEEE, 2020. 11, 14

[40] John, Meenu Mary, Helena Holmström Olsson, and Jan Bosch: *Towards MLOps: A framework and maturity model.* In *Proceedings of the 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 1–8. IEEE, 2021. 11, 14

[41] Karlaš, Bojan, Matteo Interlandi, Cedric Renggli, Wentao Wu, Ce Zhang, Deepak Mukunthu Iyappan Babu, Jordan Edwards, Chris Lauren, Andy Xu, and Markus Weimer: *Building continuous integration services for machine learning.* In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2407–2415. ACM, 2020. 11, 16

[42] Zhou, Yue, Yue Yu, and Bo Ding: *Towards MLOps: A case study of ML pipeline platform.* In *Proceedings of the International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, pages 494–500. IEEE, 2020. 12, 16

[43] Silva, Lucas Cardoso, Fernando Rezende Zagatti, Bruno Silva Sette, Lucas Nildaimon dos Santos Silva, Daniel Lucrédio, Diego Furtado Silva, and Helena de Medeiros Caseli: *Benchmarking machine learning solutions in production.* In *Proceedings of the 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 626–633. IEEE, 2020. 12

[44] Cloud, Google: *Mlops: Continuous delivery and automation pipelines in machine learning.* Google Cloud Architecture Framework, 2020. `https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning`. 14, 24, 27, 28, 63, 65

[45] Li, Y., J. Tian, A. Xu, R. Greiner, J. Hayward, A. J. Greenshaw, and B. Cao: *Maturity framework for operationalizing machine learning applications in health care: Scoping review.* Journal of Medical Internet Research, 27:e66559, 2025. 14

[46] Nahar, Nadia, Shurui Zhou, Grace Lewis, and Christian Kästner: *Collaboration challenges in building ML-enabled systems: Communication, documentation, engineering, and process.* In *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 413–425. ACM, 2022. 15, 18, 19, 58, 60, 66

[47] Wan, Zhiyuan, Xin Xia, David Lo, and Gail C. Murphy: *How does machine learning change software development practices?* IEEE Transactions on Software Engineering, 47(9):1857–1871, 2021. 15, 17, 64

[48] Granlund, Tuomas, Aleksi Kopponen, Vlad Stirbu, Lalli Myllyaho, and Tommi Mikkonen: *MLOps challenges in multi-organization setup: Experiences from two real-world cases.* In *Proceedings of the 1st IEEE/ACM Workshop on AI Engineering – Software Engineering for AI (WAIN)*, pages 82–88. IEEE, 2021. 16, 18, 20

[49] Vogelsang, Andreas and Markus Borg: *Requirements engineering for machine learning: Perspectives from data scientists.* In *Proceedings of the 27th IEEE International Requirements Engineering Conference Workshops (REW)*, pages 245–251. IEEE, 2019. 17, 64

[50] Villamizar, Hugo, Tatiana Escovedo, and Marcos Kalinowski: *Requirements engineering for machine learning: A systematic mapping study.* In *Proceedings of the 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 29–36. IEEE, 2021. 17, 18, 64

[51] Ahmad, Khlood, Mohamed Abdelrazek, Chetan Arora, Muneera Bano, and John Grundy: *Requirements engineering for artificial intelligence systems: A systematic mapping study.* Information and Software Technology, 158:107176, 2023. 17, 64

[52] Horkoff, Jennifer: *Non-functional requirements for machine learning: Challenges and new directions.* In *Proceedings of the 27th IEEE International Requirements Engineering Conference (RE)*, pages 386–391. IEEE, 2019. 17

[53] Habibullah, Khan Mohammad and Jennifer Horkoff: *Non-functional requirements for machine learning: Understanding current use and challenges in industry.* In *Proceedings of the 29th IEEE International Requirements Engineering Conference (RE)*, pages 13–23. IEEE, 2021. 17

[54] Lewis, Grace A., Stephany Bellomo, and Ipek Ozkaya: *Characterizing and detecting mismatch in machine-learning-enabled systems.* In *Proceedings of the 1st IEEE/ACM Workshop on AI Engineering – Software Engineering for AI (WAIN)*, pages 133–140. IEEE, 2021. 18, 62, 69

[55] Zarour, Mohammad, Hamza Alzabut, and Khalid T. Al-Sarayreh: *Mlops best practices, challenges and maturity models: A systematic literature review.* Information and Software Technology, 183:107733, 2025, ISSN 0950-5849. `https://www.sciencedirect.com/science/article/pii/S0950584925000722`. 19

[56] Mailach, Alina and Norbert Siegmund: *Socio-technical anti-patterns in building ml-enabled software: Insights from leaders on the forefront.* In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 690–702, 2023. 19

[57] IBM Corporation: *Ibm global ai adoption index 2022.* Technical report, IBM, 2022. `https://www.ibm.com/downloads/documents/us-en/107a02e94a48f5c1`, Retrieved February, 2024. 20

[58] Faubel, Leonhard and Klaus Schmid: *Mlops: A multiple case study in industry 4.0*, 2024. `https://arxiv.org/abs/2407.09107`. 20

[59] Sambasivan, Nithya, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo: *"everyone wants to do the model work, not the data work": Data cascades in high-stakes ai.* In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery, ISBN 9781450380966. `https://doi.org/10.1145/3411764.3445518`. 20, 62, 69

[60] Braun, Virginia and Victoria Clarke: *Using thematic analysis in psychology.* Qualitative Research in Psychology, 3(2):77–101, 2006. `https://doi.org/10.1191/1478088706qp063oa`. 29, 36, 39, 73

[61] Runeson, Per and Martin Höst: *Guidelines for conducting and reporting case study research in software engineering.* Empirical Software Engineering, 14(2):131–164, April 2009, ISSN 1573-7616. `https://doi.org/10.1007/s10664-008-9102-8`. 28, 72

[62] Kallio, Hanna, Anna Maija Pietilä, Martin Johnson, and Mari Kangasniemi: *Systematic methodological review: developing a framework for a qualitative semi-structured interview guide.* Journal of advanced nursing, 72(12):2954–2965, 2016. 32

[63] Patton, Michael Quinn: *Qualitative research & evaluation methods: Integrating theory and practice.* Sage publications, 2014. 35

[64] Ornelas, Tatiane, Allysson Allex Araújo, Júlia Araújo, Marina Araújo, Bianca Trinkenreich, and Marcos Kalinowski: *Llm-assisted thematic analysis: Opportunities, limitations, and recommendations*, 2025. https://arxiv.org/abs/2511.14528. 37, 39

[65] Zhang, Jie M, Mark Harman, Lei Ma, and Yang Liu: *Machine learning testing: Survey, landscapes and horizons.* IEEE Transactions on Software Engineering, 48(1):1–36, 2020. 60

[66] Silva, Francisco Luciano Quirino da, Andréia Libório Sampaio, Carla Ilane Moreira Bezerra, and Ingrid Teixeira Monteiro: *A process for eliciting trustworthy requirements in machine learning systems.* iSys - Brazilian Journal of Information Systems, 18(1):2:1–2:29, 2025. https://journals-sol.sbc.org.br/index.php/isys/article/view/3878. 64

[67] Lincoln, Yvonna S. and Egon G. Guba: *Naturalistic Inquiry.* SAGE Publications, Beverly Hills, CA, 1985. 72

Appendix A

Codebook

The coding scheme comprises 67 first-order codes. Following the coding process of Figure 4.1, challenges were coded *inductively* and practices *deductively* against the Alves et al. codebook [2]. The two groups are therefore reported separately: Table A.1 lists the 36 challenge and activity codes that answer RQ1, and Table A.2 the 31 practice codes that answer RQ2.

Practice identifiers encode the outcome of the deductive mapping. A practice with a clean semantic match in the Alves et al. codebook carries that codebook's identifier (P-xx, 12 codes) and is shown in Table A.2 alongside the corresponding catalog entry. A practice with no equivalent there was recorded as a new practice (NP-x, 19 codes) inductive to this study. Notably, the four coordination practices NP-4–NP-7 are all inductive, as the Alves et al. codebook has no dedicated team-structure category.

Within each table, codes are grouped by name family and each carries a short definition, the theme it is nested under in the project (see Figure 4.1), and its groundedness (number of coded quotations). Because codes are grouped analytically, a family may span themes, and a single quotation may carry more than one code — the groundedness subtotals therefore exceed the number of distinct quotations. Groundedness values were regenerated from the live project on 14 Jul 2026 and include the seven validation interviews (#I3 and #I5–#I7, Case A, and #I4, #I8 and #I9, Case B), which introduced the inductive codes DL7 and ML15 (#I3/#I4), OR8 (#I5), and DL8 (#I9). Total coded quotations: 302 (226 Case A, 76 Case B).

Table A.1: First-order challenge and activity codes (RQ1).

Code	Definition	Theme	G
DC — Data dependency & contracts (groundedness 98)			
DC1	Data format change without notification	T1	10

Table A.1 continued

Code	Definition	Theme	G
DC2	Data availability dependency blocking ML	T1	28
DC3	Missing/absent data contract	T1	18
DC4	Data quality insufficient for ML use	T1	7
DC5	Data pipeline refactoring burden	T1	11
DC6	Data sovereignty / LGPD constraint	T2	10
DC8	Data collection or extraction task	T1	14
IR — Infrastructure & regulatory (groundedness 72)			
IR1	Hardware or compute limitation	T2	20
IR2	Government cloud restriction (Dataprev)	T2	21
IR3	Third-party platform dependency	T2	7
IR5	Shared compute contention	T2	5
IR6	Infrastructure architecture decision	T2	19
OR — Roles & coordination (groundedness 66)			
OR1	Unclear role or ownership boundary	T3	16
OR2	Knowledge silo between teams	T3	32
OR3	Handoff ambiguity at deployment boundary	T3	17
OR8	Key-person dependency / single-point process bottleneck (bus factor)	T3	1
RE — Requirements engineering (groundedness 44)			
RE1	Undefined or dynamically changing requirements	T4	26
RE2	Stakeholder misalignment or late feedback	T4	8
RE4	Requirements validation deferred to post-implementation	T4	0
RE5	Model/feature design before data feasibility check	T4	10
DK — Domain knowledge (groundedness 50)			
DK1	Civic/political vocabulary building challenge	T5	12
DK2	SME dependency for label definition	T5	20
DK3	Domain taxonomy ambiguity or contestation	T5	16
DK6	Human-machine construal misalignment (semantic match vs. discrete axes)	T5	1
DK7	Medium-specific disinformation modeling	T5	1
ML — ML operationalization, monitoring & evaluation (groundedness 83)			
ML1	Missing evaluation criteria or metrics	T7	19
ML2	Model training or retraining task	T6	9
ML3	Model deployment or API creation	T6	30

Table A.1 continued

Code	Definition	Theme	G
ML4	Production monitoring absent or partial	T7	18
ML6	Model performance degradation risk	T7	5
ML15	Manual failure recovery (restart-to-last-state, no automated recovery)	T6	2
DL — Delivery (groundedness 56)			
DL1	Sprint delivery blocked by inter-team dependency	T6	29
DL5	Work-in-progress pile-up	T6	3
DL6	Integration challenge between ML and main platform	T6	21
DL7	Innovation without external delivery demand/deadline (research-mode autonomy)	T8	1
DL8	Prototype-to-production tooling migration/rework (PoC framework not production-viable)	T6	2

Table A.2: First-order practice codes (RQ2), with their Alves et al. cross-reference.

Code	Definition (this study)	Alves et al. practice	Theme	G
DC — Data dependency & contracts (groundedness 12)				
NP-1	Gold table as inter-team data contract	— *inductive*	T1	9
NP-2	Automated large-scale data ingestion pipeline	— *inductive*	T1	3
IR — Infrastructure & regulatory (groundedness 10)				
NP-3	Local open-source LLM adoption (regulation-driven)	— *inductive*	T2	10
OR — Roles & coordination (groundedness 61)				
NP-4	Cross-functional coordination mechanism adopted	— *inductive*	T3	36
NP-5	Translation work between technical/domain roles	— *inductive*	T3	14
NP-6	Monthly report as delivery accountability mechanism	— *inductive*	T8	0
NP-7	Showcase/alignment meeting as coordination practice	— *inductive*	T3	11

Table A.2 continued

Code	Definition (this study)	Alves et al. practice	Theme	G
RE — Requirements engineering (groundedness 59)				
P-10	Prototyping used as RE elicitation technique	Apply prototyping to validate and align objectives	T4	23
NP-8	Use case scoping or feature roadmapping	— *inductive*	T4	30
NP-9	Criteria calibration through practical tests	— *inductive*	T4	6
DK — Domain knowledge (groundedness 15)				
P-47	Multi-persona / multi-stakeholder labeling	Data labeling	T5	4
NP-10	Process-defined categories as ground truth	— *inductive*	T5	11
ML — ML operationalization, monitoring & evaluation (groundedness 43)				
P-26	Experiment tracking adoption (MLFlow)	Manage the experiments	T7	3
NP-11	Model benchmarking or selection	— *inductive*	T7	10
P-95	CI/CD pipeline hardening (Docker build, Secret Detection, DAG-import validation)	CI/CD	T6	5
NP-12	RAG evaluation harness integration (RAGAS)	— *inductive*	T7	1
P-90	LLM observability integration (LangFuse)	Continuous model monitoring	T7	5
P-61	Model interpretability / logic alignment	Interpretability	T6	8
NP-13	Batch human-in-the-loop training and measurement	— *inductive*	T6	7
NP-14	Real-time reviewer/revision handling	— *inductive*	T7	4
P-88	Contextual (electoral) concept drift	Build a degraded sensitive model (drift)	T7	0
DL — Delivery (groundedness 18)				
P-92	Decoupled delivery architecture adopted	Build specialized and reusable pipelines	T8	2

Table A.2 continued

Code	Definition (this study)	Alves et al. practice	Theme	G
NP-15	Time-boxing/deadline-setting for ML tasks	— *inductive*	T8	3
P-75	Agile process adapted for ML uncertainty	Use agile practices	T8	13
Additional adopted practices (groundedness 56)				
P-73	Modular/componentized pipeline design	Modularizing train code	T6	16
NP-16	Resource-constrained model selection	— *inductive*	T2	14
NP-17	N8N/Enlaight as orchestration workaround	— *inductive*	T6	6
P-83	Anonymized mirrored dev environment	Create a controlled environment	T2	0
NP-18	ICE Score for ML feature prioritization	— *inductive*	T4	1
P-22	Architecture documentation as knowledge externalization	Documentation	T3	9
NP-19	Specification artifact as pre-development contract	— *inductive*	T4	10

Appendix B

Meetings held by the Machine Learning team

Table B.1: Meetings held on Machine Learning

Code	**Date**	**Main Topics Addressed**	**Link**
#R01	Apr 29, 2025	Manual classification of proposals with IPOL experts; privacy and use of local LLM (Sabiá); planning a multidisciplinary workshop.	Link
#R02	Sep 05, 2025	Discussion on the vector database (Qdrant), use of generative AI for summaries, model accuracy (63% to 84%), and CPU vs GPU inference.	Link
#R03	Jan 23, 2026	Definition of the initial version (V0) of automated reports, use of the "Climate Plan" process for testing, and HTML cleaning.	Link
#R04	Jan 30, 2026	*Note: Content identical to R03.* Focus on automated report tests in the "Climate Plan".	Link
#R05	Feb 06, 2026	Sprint planning, extraction and cleaning of HTML from processes, engagement metrics (High-light), generation of summaries with LLMs.	Link

Code	Date	Main Topics Addressed	Avaliable
#R06	Feb 10, 2026	Product alignment, segmentation of user profiles (engaged, one-time, influencer), taxonomy and vocabulary challenges (BCGE).	Link
#R07	Feb 10, 2026	ML use cases (hate speech detection, proposal aggregation, RAG chatbot), dynamics of dictionaries vs. clustering.	Link
#R08	Feb 20, 2026	Focus on HTML cleaning, extraction, and summaries.	Link
#R09	Feb 27, 2026	Implementation of MLflow for model traceability, training with logistic regression for textual structure analysis, and heuristics.	Link
#R10	Mar 06, 2026	Hardware constraints (Mac Studio), orchestration of agents via N8n/Enlight, and review of heuristics to measure engagement.	Link
#R11	Mar 13, 2026	Text generation testing, chatbot integration via Enlight or iframes, and overcoming data blocks imposed by Oracle Infinity.	Link
#R12	Mar 17, 2026	Writing of a scientific article, recovery of requirements lost over time, and thematic analysis of the socio-technical benefits of the ML pipeline.	Link
#R13	Mar 27, 2026	Creation of "Gold" tables by Data Engineering, validation of executive report structure, and bureaucracy/architecture issues with Dataprev.	Link
#R14	Apr 10, 2026	MLflow and Airflow architecture to promote models to production, restrictions with files via MinIO and database, and data provenance.	Link

Code	Date	Main Topics Addressed	Avaliable
#R15	Apr 17, 2026	Debates on data architecture (Data Lake vs pgvector), decoupling ML deliveries via N8n/Enlight to bypass bottlenecks in the BP development cycle.	Link
#R16	N/D	Short alignment presentation on data products: Topic Analysis, Hate Speech, and Chatbot (RAG).	Link