UNIVERSITY OF CALGARY

Democratizing Software Development and Machine Learning Using Low Code Applications

by

Md Abdullah Al Alamin

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING

CALGARY, ALBERTA

DECEMBER, 2022

Abstract

Low-code software development (LCSD) is an emerging approach to democratize traditional and Machine Learning (ML) application development for practitioners from diverse backgrounds. Traditional LCSD platforms promote rapid application development with a drag-and-drop interface and minimal programming by hand. Similarly, low-code Machine Learning (ML) solutions (aka, AutoML) aim to democratize ML development to domain experts by automating many repetitive tasks in the ML pipeline (e.g., data pre-processing, feature engineering, model design, and hyper-parameter configuration). The rapid emergence of LCSD platforms warrants systematic studies to understand the challenges developers/practitioners face while using the platforms. This thesis catalogs, for the first time in the literature, the challenges developers face while using low code platforms developed for traditional and ML software application development. To the end, we also offer our hands on experience of developing a low code ML software systems for our industrial partner.

Specifically, we investigate the current status, i.e., services of LCSD providers, open-source research & collaboration. We conduct the LCSD practitioners' challenges by analyzing their discussion on the popular Q&A forum Stack Overflow (SO) to seek technical assistance. To further validate our findings, we conduct to develop a low-code machine learning solution in collaboration with domain experts from industry and academia. Additionally, we develop AutoGeoML, an open-source low-code framework that solves the current limitations of low-code ML solutions.

Our qualitative investigation of 121 traditional and 37 AutoML LCSD services shows that around 60% traditional LCSD solutions are related to business process management (BPM) and work process automation, 90% are proprietary, and only 63% platforms support only proprietary cloud deployment options. We find that around 65% of services offer shallow or general purpose ML applications, 57% solutions are open-sourced and offer flexible deployment options. According to our findings, Customization and LCSD Platform Adoption are the most discussed topic, followed by Data Management. We also find that Deployment and Maintenance are still the most difficult Software Development Life Cycle (SDLC) phase. We highlight the limitations of current low-code AutoML services and develop AutoGeoML. This open-source low-code framework provides domain-experts-in-the-loop customizability, and modular abstraction is some of the critical requirements lacking in existing AutoML frameworks.

The findings of this thesis have implications for all three LCSD stakeholders: LCSD platform vendors, LCSD practitioners, and Researchers. Researchers and LCSD solution vendors can collaborate to improve different aspects of LCSD, such as better tutorial-based documentation, DevOps support, and expert-in-the-loop customizability.

Preface

This thesis is an original work by the author and written in "manuscript-based" style. The following is a list of published papers that are on topic of this thesis:

- Md Abdullah Al Alamin, Sanjoy Malakar, Gias Uddin, Sadia Afroz, Tameem Haider, and Anindya Iqbal. An empirical study of developer discussions on low-code software development challenges. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pages 46–57. IEEE, 2021. (Chapter 3).
- Md Abdullah Al Alamin, Gias Uddin, Sanjay Malakar, , Sadia Afroz, Tameem Haider, and Anindya Iqbal. Developer discussion topics on the adoption and barriers of low code software development platforms. Empirical Software Engineering, 28(1):1–59, 2023. (Chapter 3).
- Md Abdullah Al Alamin and Gias Uddin. Quality assurance challenges for machine learning software applications during software development life cycle phases. In 2021 IEEE International Conference on Autonomous Systems (ICAS), pages 1–5. IEEE, 2021. (Partially in Chapter 4 and 5)

The following is a list of papers that are currently under review and are on topic of this thesis:

- Md Abdullah Al Alamin and Gias Uddin. "Challenges and Barriers of Using Low Code Software for Machine Learning" in ACM Transactions on Software Engineering and Methodology (TOSEM) journal in 2022. (Chapter 4)

Acknowledgments

This thesis was only possible with the support and guidance of my fantastic supervisor, Gias Uddin. We had some excellent experiences working together, passionately discussing different research approaches. Finally, with extreme dedication and hard work, we delivered on time. My technical writing and research approaches are learned from Dr Gias. I will never forget his support during our MSR 2021 conference paper submission, where he significantly contributed and provided me with hands-on guidance. I am fortunate to have you as my supervisor, and I hope you will always be my mentor.

I am also grateful to Dr Anindya Iqbal and Dr Shohrab Hossain from the Bangladesh University of Engineering and Technology for supporting me when I needed it. I had the opportunity to research with Dr Anindya, which has been a crucial part of this dissertation. He has always been there to guide me and answer my questions. I also thank Sanjay Malakar, Sadia Afroz, and Tameem Bin Haider for their contribution in empirical analysis for publications that are part of this dissertation. I thank all the human coders who participated in the data analysis of this thesis. I am grateful for the support and guidance from Dr Roman Shor, whose support and guidance enabled me to conduct an industrial evaluation for this thesis. I especially thank Dr Morshedul Islam, who has supported me academically and personally.

I am grateful to all of my friends in academia and industry who have been essential to my research. I thank all the members of DISA Lab who have supported me present and organizing my research thoughts. I am especially grateful for having Ajoy Das and Junaed Younus Khan, who has been a reliable source of support, especially during the uncertain period of Covid-19. I thank open-source software developers for sharing their research data for analyzing Stack Overflow discussions. I also thank my professors from the University of Calgary, who has been supportive during my course works and research projects.

Finally, my hearty gratitude towards my wife, who has supported me throughout this period. I can never overestimate the personal sacrifices and compromises she made to support me. I am also grateful to my parents for the sacrifices that they made so that I could make progress in my career. It took a great deal of patience and sacrifices from my family along this journey.

Dedication.

To the three most significant people in my life: 1) My father, who has gone above and above throughout my life to inspire and support me. Almost certainly, I would not be where I am now without his sacrifices and efforts. 2) Tanjina Sharmin, my wife, who has been a constant source of love, support, and encouragement. I am grateful to have you in my life. 3) My mother, whose unconditional love has always been a reliable resource of emotional support.

Table of Contents

Abstract ii

Preface iii

Acknowledgments iv

Dedication v

Table of Contents viii

List of Figures x

List of Tables xi

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Overview 2
- 1.3 Objective and Scope 3
- 1.4 Contributions 4
- 1.5 Thesis Organization 5

2 An Exploratory Study of Different Types of Low-code Software Applications 6
- 2.1 Overview of LCSD 7
  - 2.1.1 Low-code in traditional software development 9
  - 2.1.2 Low-code in machine learning development 15
- 2.2 Characterization of Traditional Low-code platforms 18
  - 2.2.1 Motivation 18
  - 2.2.2 Approach 18
  - 2.2.3 Result 20
- 2.3 Characterization of Low-code ML tools/platforms 23
  - 2.3.1 Motivation 23
  - 2.3.2 Approach 24
  - 2.3.3 Result 25
- 2.4 Summary 29

3 Challenges and Barriers of Using Low Code Software for Traditional Application Development 31
- 3.1 Introduction 32
- 3.2 Background 34
- 3.3 Study Data Collection and Topic Modeling 36
  - 3.3.1 Data Collection 36
  - 3.3.2 Topic Modeling 40
- 3.4 Empirical Study 41
  - 3.4.1 What topics are discussed about LCSD in Stack Overflow? (RQ1) 42
  - 3.4.2 How do the LCSD topics evolve over time? (RQ2) 51

3.4.3 What types of questions are asked across the observed topic categories? (RQ3) 55
3.4.4 How are the observed topic categories discussed across SDLC phases? (RQ4) 59
3.4.5 What LCSD topics are the most difficult to get an accepted answer? (RQ5) 64
3.5 Discussions 70
3.5.1 Issues with not accepted answers or posts with negative score 70
3.5.2 Discontinued low-code platforms and future trends 71
3.5.3 LDA parameter Analysis 72
3.5.4 The Prevalence & Evolution of Top Ten LCSD platforms 72
3.5.5 The Case of "aggregating" data across multiple LCSD platforms 74
3.5.6 The application of the taxonomy of challenges 75
3.5.7 Implications 75
3.6 Threats to Validity 81
3.7 Related Work 82
3.7.1 Research on Low Code Software Development and Methodologies 82
3.7.2 Topic Modeling in Software Engineering 83
3.8 Summary 85

4 Challenges and Barriers of Using Low Code Software for Machine Learning Application Development 86
4.1 Introduction 88
4.2 Background 90
4.2.1 AutoML as Low-code Tool/Platform for ML 90
4.2.2 AutoML Approaches 92
4.2.3 AutoML Services 92
4.3 Study Data Collection and Topic Modeling 93
4.3.1 Data Collection 93
4.3.2 Topic Modeling 96
4.4 Empirical Study 97
4.4.1 RQ1. What topics do practitioners discuss regarding AutoML services on SO? 97
4.4.2 RQ2. How AutoML topics are distributed across different machine learning life cycle phases? 103
4.4.3 RQ3. What AutoML topics are most popular and difficult on SO? 106
4.4.4 RQ4. How does the topic distribution between cloud and non-cloud AutoML services differ? . 111
4.5 Discussions 115
4.5.1 Evolution of AutoML related discussion 115
4.5.2 Top AutoML service providers 118
4.5.3 Traditional ML vs AutoML challenges 119
4.5.4 Traditional vs. AutoML LCSD practitioners challenges 122
4.5.5 The application of the taxonomy of challenges 124
4.6 Implications 124
4.7 Threats to Validity 126
4.8 Related Work 127
4.8.1 Research on AutoML 127
4.8.2 Empirical Study on Challenges of ML Applications 127
4.8.3 Research on Topic Modeling & SO discussion 128
4.9 Summary 129

5 AutoGeoML: A Low Code Machine Learning Toolkit for Geothermal Software 130
5.1 Introduction 131
5.2 Related Work 132
5.3 Background 133
5.3.1 Vibration Forecasting 133
5.3.2 Requirement collection & Scope definition 134
5.4 AutoGeoML 137
5.4.1 Design Motivation 138
5.4.2 Design Overview 142

5.4.3 API description . . . 143
5.5 Discussions . . . 145
5.5.1 AutoML challenges during AutoGeoML development . . . 146
5.6 Usefulness of the findings for Project Managers . . . 148
5.7 Summary . . . 149

6 Conclusion and Future Work 150
6.1 Summary . . . 150
6.2 Limitations and Future Work . . . 151

Bibliography 152

List of Figures

2.1 An overview of research on the low-code approach to democratizing software development. 8
2.2 A screenshot of a WYSIWYG (what you see is what you get) interface to aid with UI design [270]. . 9
2.3 An screenshot of Salesforce BPMN notation for implementing business logic (Workflow automation) [56]. 10
2.4 A screenshot of the drag-and-drop graphical user interface for the open-source LCSD platform in a local development setting. [138]. 10
2.5 Agile methodologies in traditional vs. LCSD development 13
2.6 An overview of AutoML for discovering the most effective model through Neural Architecture Search (NAS) [31] 15
2.7 An overview of traditional ML pipeline vs. AutoML tools/platform pipeline. 16
2.8 A high-level overview of Google Cloud AutoML solution [1] 17
2.9 A distribution of different types of applications supported by LCSD platforms. 19
2.11 Distribution of different types of ML application development solutions offered by AutoML vendors. 25

3.1 Agile methodologies in traditional vs LCSD development 35
3.2 Distribution of LCSD related recommended vs relevant tags based on different u and V values from our initial taglist. 37
3.3 Distribution of Questions and Topics per Topic Category 43
3.4 Distribution of questions by low-code topics by total number of posts (Number of Questions + Number of Accepted Answers) 44
3.5 A taxonomy of LCSD topics with sub-categories. 45
3.6 The evolution of overall LCSD-related discussions over time. 52
3.7 The absolute impact for LCSD topic categories over time. 53
3.8 The relative impact for LCSD topic categories over time. 53
3.9 Distribution of different question Types 57
3.10 Distribution of questions (Q) per SDLC phase 60
3.11 The popularity vs. difficulty of getting an accepted answer for LCSD Topic categories. 66
3.12 SO questions with a negative score or unaccepted SO answer. 71
3.13 The different coherence values for varied K on different runs. 72
3.14 The evolution of top ten LCSD platforms discussions over time. 73
3.15 The distribution of topic categories across top ten LCSD platforms. 74
3.16 The percentage of questions distributed across five topic categories for the top ten LCSD platforms versus the rest of the platforms. 74
3.17 The popularity vs. difficulty bubble chart for 40 LCSD-related topics. 76
3.18 The popularity vs. difficulty bubble chart for low-code software development life cycle phases 79
3.19 The popularity vs. difficulty of different types of LCSD-related questions 80

4.1 An overview of AutoML for discovering the most effective model through Neural Architecture Search (NAS) [31] 91
4.2 An overview of traditional ML pipeline vs AutoML pipeline. 92
4.3 Distribution of Questions and Topics per Topic Category. 99
4.4 Distribution of posts (#Questions + #Accepted Answers) of AutoML topics. 99
4.5 A taxonomy of 13 AutoML topics organized into four categories. 100

4.6 Distribution of questions (Q) per MLLC phase 104
4.7 The popularity vs. difficulty of AutoML topic categories. 108
4.8 Distribution of questions on cloud vs non-cloud AutoML services. 112
4.9 The distribution of AutoML topic categories between cloud vs non-cloud AutoML services. 113
4.10 An overview of combined difficulty and popularity between cloud vs non-cloud AutoML solution's topic categories. 114
4.11 The popularity vs. difficulty for AutoML pipeline phases. 115
4.12 The popularity vs. difficulty for AutoML topics. 116
4.13 The evolution of overall AutoML related questions over time. 117
4.14 The evolution of SO questions on AutoML topic categories over time. 118
4.15 The evolution of questions for top AutoML services. 119
4.16 The status of top platforms and their percentage of accepted answers. 120
4.17 An example of H2O's platform's support for resolving reported bugs ($A_{51683851}$). 120
4.18 An example of Amazon Sagemaker team monitoring and supporting SO practitioners' queries ($A_{55553190}$). 121

5.1 A high-level overview of automation requirements for different ML pipelines. 134
5.2 An high level overview of human-in-the-loop Machine learning [269]. 141
5.3 A high level overview of human-in-the-loop AutoGeoML architecture. 142
5.4 A high-level overview of AutoGeoML API modules for different tasks. 143
5.5 A screenshot of sample API usage for AutoGeoML. 144

List of Tables

3.1 Types of questions across the LCSD topic categories . . . 57
3.2 Distribution (frequency) of LCSD topics per SDLC phase. Each colored bar denotes a phase (Black = Requirement Analysis, Green = Application Design, Magenta = Implementation, Red = Testing, Blue = Deployment, Orange = Maintenance) . . . 62
3.3 Types of questions across the Software development life cycle phases . . . 63
3.4 Popularity for getting an accepted answer for LCSD topics . . . 67
3.5 Difficulty for getting an accepted answer for LCSD topics . . . 68
3.6 Correlation between the topic popularity and difficulty . . . 69
3.7 The summary of our core findings and recommendations . . . 77
3.8 Comparing the popularity and difficulty metrics of different domains . . . 83
3.9 Comparative analysis of question types across different domains . . . 84

4.1 Distribution (frequency) of AutoML topics per MLLC phase. Each colored bar denotes a phase (Black = Requirement Analysis, Green = Data Preparation, Magenta = Model Designing, Red = Training, Blue = Model Evaluation, Orange = Model Deployment & Monitoring) . . . 104
4.2 Popularity metrics for AutoML topics. . . . 109
4.3 Difficulty for getting an accepted answer for AutoML topics . . . 109
4.4 Correlation between the topic popularity and difficulty. . . . 110
4.5 Distribution (frequency) of MLLC phase between cloud vs non-cloud AutoML services. Each colored bar denotes a phase (Black = Cloud, Green = Non-cloud Solution) . . . 113
4.6 The comparison between the struggles of traditional LCSD practitioners and AutoML practitioners. . 123

5.1 The summary of the requirements of the system . . . 134
5.2 The summary observed AutoML related topics during AutoGeoML development . . . 147
5.3 The summary of AutoML challenges observed in SO discussion vs AutoGeoML development . . . 148

Chapter 1

Introduction

1.1 Motivation

Software development is a structured process of specifying, coding, debugging, testing and documenting the final manifestation of the software [227]. Software engineering is always evolving, and one of the recent trends is that more software applications are developed in collaboration with software developers and domain experts. The growing need for cost-effective, highly functional software with short development time has caused the rise of a new paradigm of software development called LCSD. The shortage of skilled professional developers and the improvement of different automation tools and cloud computing play a critical role in its mass adaptation. Demand for software developers is high, but the supply for qualified developers is low [238]. There is a huge talent shortage in the software industry, and by 2026 around 85.2 Million jobs can go unfulfilled reported by the U.S. Bureau of Labor Statistics [149].

A business may have a specific business requirement, and the stakeholder associated with that domain are aware of this, but they lack the skills to develop a software application. Generally, there are primarily two key users for a software project.

1. **Domain Experts** are knowledgeable about the problem domain (e.g., HR management, hotel management, rain forecasting, cancer diagnosis, etc.). They are either the primary users of the system (i.e., internal applications) or service providers (i.e., e-commerce site, online hotel booking system). So they have the best understanding of the system requirements, end users, dataset, etc. They decide which features to prioritize based on the business/user requirement.

2. **Technical Experts** are experienced with the nuances of overall software development, which requires a rigorous process of designing, coding, testing, and deploying the application. It requires a wide range of skill-set from understanding underlying hardware, operating system, networking, web protocol, and a wide range of program-

ming languages. In the ML context, they understand the nuances of the ML model. They have a greater understanding and experience in selecting the appropriate ML algorithm/architecture, engineering features, training the model, and evaluating its performance.

The primary objective of low-code services is to bridge the gap between the business requirement and the developers' skill constraints, which is a common reason for long development times in complex business applications. This is necessary since a business may have a specific business requirement, and the domain experts/stakeholders in the business know it. Still, they lack the skills to develop an ML application.

1.2 Overview

The overall LCSD approach can be divided into two types: (1) Low-code in traditional software development: refers to applications often written using general-purpose programming languages (such as C++, Java, Python, PHP, etc.) or Web technologies. (2) Low-code in ML software development: describes the development of data analytics/machine learning applications using labelled datasets with minimum coding.

Low-code in traditional software development. One of the main motivations for adopting a low-code approach is professional software developers lack domain knowledge to understand the intended use cases of the system fully. On the other hand, business people have difficulties expressing requirements. Another problem is traditional software development life cycle is too slow to meet changing business requirements. This is especially true for a startup company whose business requirement is constantly changing based on the user's feedback and market demand. LCSD has a set of strategic tools that helps to drive innovation in this ever-changing world. The demand for enterprise applications is rising, and the traditional IT department can not fulfil the demand of a massive backlog of requests. LCSD aims to accelerate digital transformation and technical innovation by engaging various domain experts from different backgrounds in software development tasks. Low-code approach is attracting more and more businesses. According to Forrester report [216], the LCSD platform market is expected to be $21 Billon by 2022. According to a Gartner report, by 2024, around 65% of large enterprises will use low-code platforms to some extent [258, 266]. All these low-code approaches help automate different complex workflow services from requirement to implementation to integration to maintenance.

Low-code in ML software development. Machine learning algorithms uncover key insights from data by training algorithms to make predictions and classifications. These insights drive decision-making in business and governance. But since this requires advanced statistical skills, ML algorithms, and computational challenges, it requires skilled professionals from multiple domains to develop such solutions. ML and data analytics insights have become quite an essential component for decision-making part for many businesses. However, an efficient ML application requires highly specialized ML experts with domain experts. The main objective of low-code ML solutions is to decrease the dependency on ML experts by providing the necessary tools to domain experts to develop a machine learning model without considerable expertise in statistics and ML algorithms. Companies recognize this potential so that the adoption rate will rise over the next few years. The rapid progress in machine learning is a great example of how low-code software development can be useful. Many companies have the necessary data that can be used to develop meaningful business applications. The evolution of the digital business model is pushing businesses to adopt ML-based services (e.g., smart chatbot, business analytics, sales forecasting, etc) into their existing operations. But most of these companies do not have in-house technical experts in machine learning. Automated machine learning (i.e., AutoML) comes to the rescue in this regard. AutoML aims to make ML more accessible to domain experts by providing end-to-end abstraction from data filtering to model designing & training to model deployment & monitoring. AutoML is becoming an essential topic for software engineering researchers as more and more AutoML applications are developed and deployed. AutoML research is progressing rapidly as academia and industry work collaboratively by making ML research their priority. For example, currently, in at least some experimental settings, AutoML tools yield better results than manually designed models by ML experts [167].

1.3 Objective and Scope

There are more than 400 traditional and ML LCSD platforms [239] offered by almost all major companies like Google [113], Google AutoML [114], Salesforce [218]. Naturally, LCSD has some unique challenges [217]. The wrong choice of LCSD application/platform may cause a waste of time and resources. Several research has been conducted to analyze SO posts (e.g., IoT [254], big data [37], blockchain [260] concurrency [7], microservices [41]). Similarly, related studies [131] show that machine learning developers find model training and hyperparameter tuning challenging, and many of the issues in these categories are due to simple coding issues. Developers make simple coding mistakes such as improper use of framework APIs, Out of GPU memory, the path not found, and library not found that results in failure in the submitted jobs in training [279]. AutoML tools can streamline all of these tasks and make machine learning development less error-prone. Companies face many software engineering challenges when they try to integrate a machine learning-based feature into their products [166]. However, these studies did not analyze discussions about LCSD solutions for traditional or ML applications.

This thesis explores the evolution, current status, and massive potential of low-code software development. This research aims to study and characterize LCSD services and their features, strengths, and weaknesses. The goal is to provide valuable insight into the current state of low-code application development technologies. The objective is to provide useful insight into the current state and opportunities for future development for researchers, software developers, low-code providers, and the OSS community. This thesis explores the current LCSD solutions for traditional and ML applications, reports the current challenges by analyzing LCSD practitioners' discussion on SO, and provides recommendations for future improvement.

In this thesis, the main research question that we explore is "what are the challenges that low-code practitioners face when they use low-code platforms?" To that end, we explore the following three sub research questions.

RQ1: What are the challenges the low-code practitioners face while developing general software applications? (Chapter 3)

We find that low-code practitioners struggle with the rigidity of the UI and application customizability. They have difficulty utilizing the functionalities offered by LCSD platforms. Despite a lot of support, application deployment and maintenance still remain the most challenging SDLC phase for low-code practitioners.

RQ2: What are the challenges the low-code practitioners face while developing machine learning software applications? (Chapter 4)

We find that MLOPs (i.e., the operationality of model development and its deployment), Model Design related topics are challenging to AutoML practitioners. Model Deployment & Monitoring and Model Evaluation are the most popular topics for AutoML solutions. AutoML practitioners find Model Deployment and Maintenance aspect most challenging.

RQ3: Do the challenges differ when we develop a low code toolkit for machine learning to support industrial applications? (Chapter 5)

During our industrial case study, we discover that current non-cloud-based AutoML solutions are viable options for end-to-end automation. However, they are significantly computationally expensive, lacks customization options and model interpretability for domain experts. But for industrial adoption, expert-in-the-loop customization and model interpretability is required. So, we develop AutoGeoML, a low-code machine learning toolbox for the geothermal domain. During ML workflow automation, this utilizes a modular design and domain-specific expertise.

1.4 Contributions

The contribution of this thesis is as follows:

1. introduces LCSD technologies, related research works, and characterization of low-code solution approach for traditional and machine learning software applications.

2. provides an in-depth analysis of practitioners' challenges with low-code services for traditional software,

3. provides an in-depth analysis of practitioners' challenges with low-code services for ML applications,

4. presents a case study of current limitations and future research direction by developing a low-code solution for horizontal drilling operations.

The findings can enhance our understanding of the developers' struggle for traditional and ML application development using LCSD solutions. The findings would help the research community and LCSD vendors better focus on the specific topics or more effective and usable tools. The practitioners can prepare for difficult topics and SDLC phases. All stakeholders can collaborate to provide enhanced documentation assistance.

1.5 Thesis Organization

The thesis is organized as follows. Chapter 2 provides background and related work for this thesis. The background work provides a high-level overview of cloud and non-cloud-based low-code solutions and traditional and machine learning applications. The related work provides an in-depth survey of current research and technologies that empower LCSD and AutoML services.

Chapter 3 presents an in-depth analysis of current practitioners' challenges of low-code solutions for traditional software development by analyzing a growing number of SO posts (i.e., 33K SO posts (questions + accepted answers)) and provides guidance for LCSD platform vendors, LCSD developers/practitioners, Researchers, and Educators. We find that Customization, Data Storage, Platform Adoption, and Third-party Integration are the most discussed topics. We also find that LCSD practitioners find the application deployment and maintenance phase most challenging.

Chapter 4 presents a detailed analysis of the current challenges and limitations for low-code machine learning applications by analyzing related 14k SO posts (questions + accepted answers) and providing guidance for future improvements. It also highlights the difference between the challenges of traditional vs machine learning low-code solutions. We find that MLOps, Models, Data, and Documentation are the most discussed topics among AutoML practitioners.

Chapter 5 presents a case study of the strength and weaknesses of low-code ML tools. It validates the prevalence of the AutoML services described in the previous Chapter. It highlights the importance of the involvement of the domain experts' role in ML workflow automation. It provides future research guidelines for researchers for the future research direction of the low-code machine learning tools. We find that end-to-end automation is neither feasible nor desirable. Rather expert-in-the-loop AutoML solutions for each ML pipeline step, with an emphasis on model interpretability, are preferable.

Chapter 6 concludes the research by summarizing the findings of the thesis, outlining the limitations and several future research directions.

Replication Package: The code and the datasets generated during this thesis is available in our public GitHub repository. `https://github.com/disa-lab/Thesis-Alamin-MSc/`

Chapter 2

An Exploratory Study of Different Types of Low-code Software Applications

As stated in Chapter 1, the low-code approach is gaining popularity across industries for both traditional and machine-learning applications. The purpose of this Chapter is to provide an overview of LCSD. In this Chapter, we characterize the overall LCSD into two types: traditional and machine-learning software development solutions. We present a comprehensive overview of the technologies that have enabled LCSD. In addition, we undertake a qualitative analysis of 121 traditional LCSD platforms and 37 ML LCSD services to determine what types of application development, licensing, and deployment flexibility they provide. The findings of this study provide a comprehensive overview of the present status of global LCSD services. At the end of this review, we discuss this domain's current and important challenges and set the stage for Chapters 3, 4, and 5.

2.1 Overview of LCSD

Software development is a process of expressing ideas into a usable product that can be used to solve certain tasks. So, in most cases, our inability to develop a suitable solution for any domain is mostly limited by our imagination. LCSD approach takes a key characteristic motivation from MDD [220], and that is it allows to consider software a concept rather than computer programs. One of the key advantages of this model-driven approach is that a concept is more relatable to the problem domain, so more domain experts can understand, maintain and participate in development easily. Programs can be generated from these models, and the limitation of programming language does not limit it. At the heart of LCSD is MDD [57], which offers design abstraction and automation.

Similarly, developing an ML model requires significant human expertise and intuition based on experience in the machine-learning context. This low-code machine learning solution seeks to solve this issue by automating some ML pipeline processes and offering a higher level of abstraction over the complexities of ML hyperparameter tuning, allowing domain experts to design ML applications without extensive ML expertise. It significantly increases productivity for machine learning practitioners, researchers, and data scientists. The primary goal of low-code AutoML tools is to reduce the manual efforts of different ML pipelines, thus accelerating their development and deployment.

With the introduction of cloud computing, there has been a greater push for software development to be adopted on a larger scale. In recent years, the low-code technique has become increasingly popular. The *no-code* software development methodology aspires to go one step farther than the low-code software development methodology. It provides a user-friendly, simple-to-use interface for designing applications and implementing business logic without writing code. Therefore, no-code is a subset of the low-code approach, where practitioners are required to write minimal code. It seeks to assist *citizen developers and citizen data scientists* who are not experts in software development or data science but are experts in their respective fields. The objective of the LCSD tools is to equip these citizen developers and domain experts with the required tools to do activities previously achievable only by professional software developers and data scientists.

A business may have a specific business requirement, and the stakeholder associated with that domain are aware of this, but they lack the skills to develop a software application. Generally, there are primarily two key users for a software project.

1. **Domain Experts** are knowledgeable about the problem domain (e.g., HR management, hotel management, rain forecasting, cancer diagnosis, etc.). They are either the primary users of the system (i.e., internal applications) or service providers (i.e., e-commerce site, online hotel booking system). So they have the best understanding of the system requirements, end users, dataset, etc. They decide which features to prioritize based on the business/user requirement.

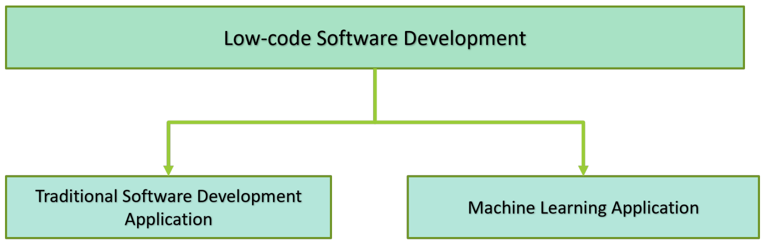

Low-code Software Development
Traditional Software Development Application
Machine Learning Application

Figure (2.1) An overview of research on the low-code approach to democratizing software development.

2. **Technical Experts** are experienced with the nuances of overall software development, which requires a rigorous process of designing, coding, testing, and deploying the application. It requires a wide range of skill-set from understanding underlying hardware, operating system, networking, web protocol, and a wide range of programming languages. In the ML context, they understand the nuances of the ML model. They have a greater understanding and experience in selecting the appropriate ML algorithm/architecture, engineering features, training the model, and evaluating its performance.

In Figure 2.1, we present a high-level overview of the LCSD approach. Initially, they are divided into two types:

1. Low-code in traditional software development: it refers to applications traditionally developed via general-purpose programming languages (e.g., C++, Java, Python, PHP, etc.) or Web technologies. A wide range of applications can be developed using them, such as customer relationship management, sales management, scheduling, etc.

2. Low-code in ML software development: it refers to the development of data analytics/machine learning applications with minimal coding and from labeled datasets only. Various applications such as data visualization, sales prediction, image/object classification, and sentiment detection are developed using this approach.

Later we categorize each of these low-code approaches into tools and cloud platforms.

1. Non-cloud-based low-code tools (library/frameworks): It is a collection of libraries, software development toolkits, and other components. They offer ready-made components for the developers to use throughout the development process. They offer better flexibility in terms of software planning to application deployment. Some of the popular low-code tools are Rintagi [211], Joget [137], AutoKeras [136], AutoWeka [146].

2. Cloud-based low-code platforms: A platform is a collection of the platform's hardware infrastructure and the software components combined with support throughout app development, i.e., from the development environment to deployment support. They offer more streamlined support, especially for application scalability, security, and maintenance. Some of the popular low-code platforms are Appian [21], Oracle Apex [18], Amazon SageMaker [35], H2O AI [119].

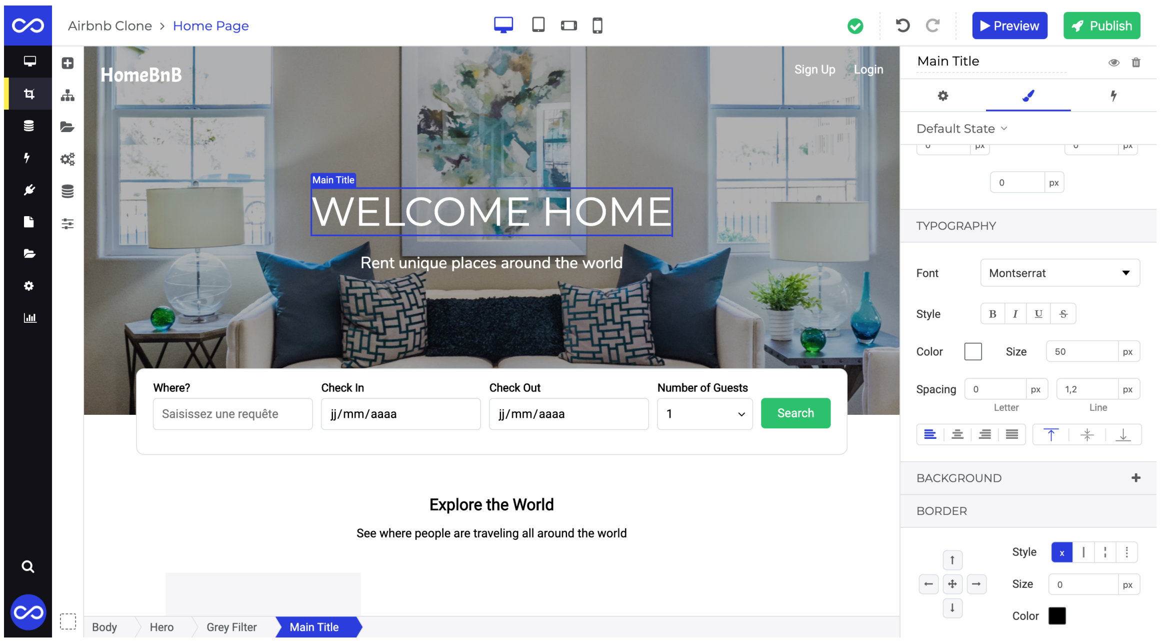


Figure (2.2)   A screenshot of a WYSIWYG (what you see is what you get) interface to aid with UI design [270].

Non-cloud-based low-code tools provide the necessary library to attain these goals in a more customizable local setup. On the other hand, low-code cloud platforms provide end-to-end support, especially in application deployment and monitoring. *We find that all the traditional LCSD services fall under the platforms category rather than tools because even the open-source traditional LCSD services (e.g., Joget [137], Skyve [226]) provide some visual interface for UI design via IDE or web interface in local environment setting (Fig. 2.4).* On the other hand, we find that a significant proportion of AutoML services fit under the tool category, as they only provide high-level programming APIs to automate ML workflows (e.g., data pre-processing, feature engineering, model design, and training). We provide a detailed analysis current state of low-code solutions for traditional (Section 2.2) and machine learning applications (Section 2.3).

2.1.1 Low-code in traditional software development

Low-code Software Application. To cater to the demand of the competitive market, business organizations often need to develop and deliver customer-facing applications quickly. LCSD platform quickly translates the business requirement into a usable software application. It also enables citizen developers of varying software development experience to develop applications using visual tools to design the user interface in a drag-and-drop manner and deploy them easily [164]. LCSD is inspired by the model-driven software principle where abstract representations of the knowledge and activities drive the development rather than focusing on algorithmic computation [217]. LCSD platforms aim to remove the testing, deployment, and maintenance complexity observed in traditional software development. In simple terms, it can be said that low-code is a visual tool to replace some handing written code. It incorporates the

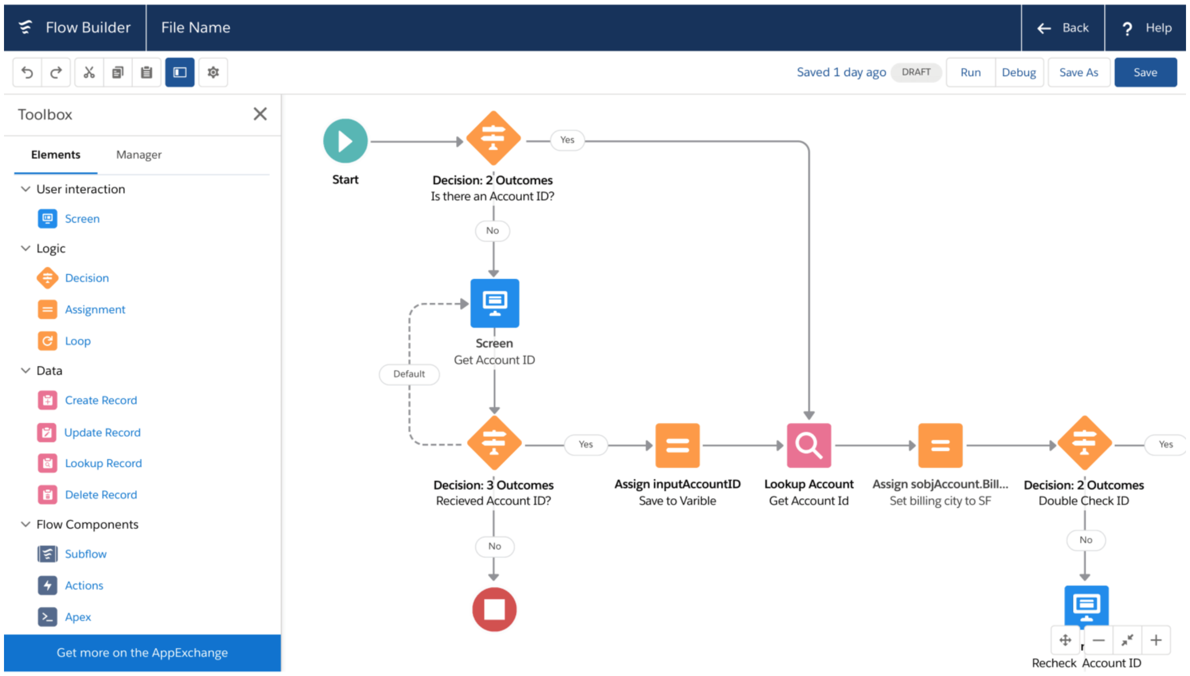


Figure (2.3) An screenshot of Salesforce BPMN notation for implementing business logic (Workflow automation) [56].

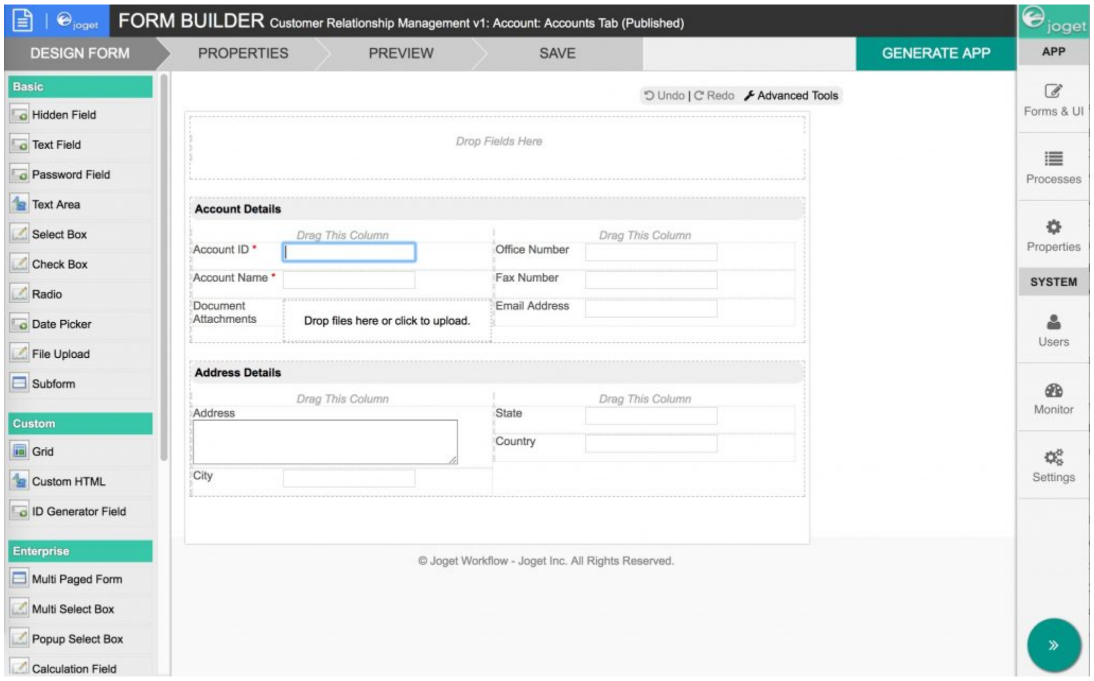


Figure (2.4) A screenshot of the drag-and-drop graphical user interface for the open-source LCSD platform in a local development setting. [138].

WYSIWYG (What You See Is What You Get) approach (Fig. 2.2) to design UI and implement business functionality via domain-specific notation (Fig. 2.3). Some of the most popular low-code platforms are Appian [21], Google App Maker [113], Microsoft Powerapps [203], and Salesforce Lightning [218].

2.1.1.1 Technologies that Shaped LCSD

A wide range of technologies aims to empower domain experts, i.e., citizen developers. In this section, we provide an overview of related technologies and terminologies. These relevant technologies are necessary to understand this study.

End User Development (EUD) is a collection of techniques and tools that enables non-professional software developers, i.e., end users, to create, modify some software artifact to some extend [69, 159, 193]. EUD [103] activities range from customizing a software application to changing the configuration to programming. EUD is closely associated with end-user programming (EUP) [139]. The fundamental distinction between EUD and LCSD is that citizen developers in LCSD wish to develop an application rapidly and without writing a great deal of code by hand. Typically, the developed application is targeted broader end users. In contrast, in the EUD paradigm, the end-users wish to customize/create an application to meet some of their specific needs. According to Fischer et al. [103], the future success of EUD hinges on building tools that end users are encouraged to learn and use in their daily work routines, and low-code has come a long way in this regard.

Model Driven Development (MDD) is a novel approach to software development where models are designed before the application code is written [25, 220]. It utilizes different modeling techniques so that the developers can develop and design applications with high-level abstraction [121]. This abstraction helps to visualize and understand complex problems and potential solutions. It simplifies software development tasks and provides structure to store different software development-related design decisions in a common vocabulary that happens naturally from requirement analysis to software development to testing and deployment phases. LCSD adopts this MDD with an agile approach, also known as agile model-driven development (AMDD), where the models are designed in an iterative approach and then fine-tuned. This model-driven approach is more suited for the citizen developers to implement the functionalities of the application visually.

Domain Specific Language (DSL) is language designed for a specific application domain [172,257]. It allows domain experts in a particular field to develop applications with the concepts he is familiar with. For example, Structured Query Language (SQL) is a DSL for interacting with relational databases. Domain-specific model language (DSML) is based on the graphical representation that allows domain experts to express domain-specific concepts easily. UML provides necessary tools that can be used for designing DSML [221]. As one of the main objectives of the LCSD platforms is to provide the necessary environment for domain experts to involve in the development process, this DSML plays a significant role in that [144]. LCSD platforms provide different DSL based on what type of target application is built using that platform. For example, many LCSD platforms such as Appian [21], Mendix [171], and Salesforce [218] offer well-known DSML called Business Process Model and Notation (BPMN) that provides graphical representations for specifying business process which is similar to activity diagram in UML (Fig. 2.3).

Visual programming is a technique that allows an end user to describe a work process by illustration rather than a typical text-based programming [54]. This type of visual presentation makes the most sense to humans. It provides an abstraction between human thinking and computers' inner workings. In a traditional software development approach, a developer knows how a computer works and designs an application accordingly. High-level programming languages and modern frameworks help to achieve that to a great extent. One of the fundamental differences between this approach and visual programming is to think of a system more naturally. It allows the creation of different classes and actors and how they define how these objects should behave at different workflows but within writing by them in a text editor. One of the drawbacks of developing general-purpose software using a pure visual programming approach is that the real-world requirement is too complex to model visually completely. To incorporate all the complex workflows, visual programming diagram becomes too complex to understand and maintain. It requires a great deal of creativity from the designer to think and design the workflows as this requires much planning, and it does not suit well with the agile approach. LCSD takes the best from visual programming and text-based programming. It allows the design of various components using visual tools and also allows writing functionalities via some degree of coding. Pure visual programming is a "no-code" approach, but low-code adaptation allows the design of various components easily and implements complex functionality.

Natural language programming is an approach to leverages recent advancements in NLP technologies [76, 158, 158, 175] to generate programming code from a natural language such as English, French. Many LCSD platforms are adopting machine learning approaches to help developers develop applications with natural language. Recently Microsoft's low-code platform powerapps will allow its citizen developers to design and develop applications using natural language [173]. This aims to provide a simple and flexible way to develop applications. Recent advancement in OpenAI's Codex [26, 62], which is based on GPT-3 [59] demonstrates outstanding performance from the software community via GitHub Co-pilot [111].

Trigger action programming and **Programming by example** are two of widely used techniques in EUD. In Trigger action programming, end users can specify some system behavior as a trigger event and define a corresponding action to be taken [256]. Programming by example is a demonstrational programming technique where an end-user can teach a computer by demonstrating some sample examples. LCSD platforms (e.g., Oracle Apex, Appian) adopt these techniques as dynamic actions. It is a graphical visualization tool for low-code practitioners to implement custom business logic based on conditional user actions or internal events.

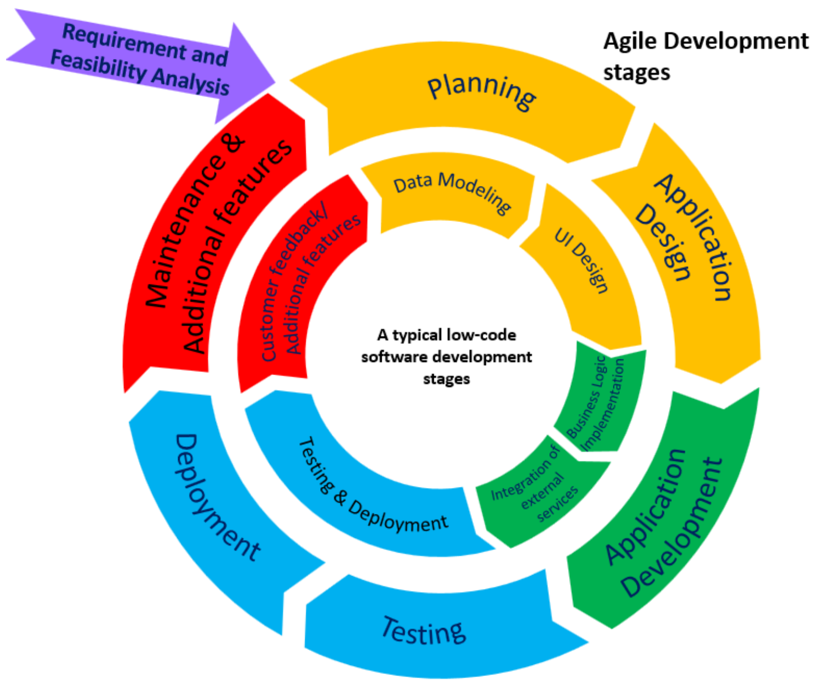

Requirement and Feasibility Analysis
Agile Development stages
Planning
Application Design
Application Development
Testing
Deployment
Maintenance & Additional features
Data Modeling
UI Design
Business Logic implementation
Integration of external services
Testing & Deployment
Customer feedback/ Additional features
A typical low-code software development stages

Figure (2.5) Agile methodologies in traditional vs. LCSD development

Serverless vs low-code software development Both LCSD and serverless approach to developing application aims to increase speedy software delivery. The serverless approach tries to minimize the complexities of server maintenance, deployment, and scalability issues. It allows developers to focus on implementing the application API without worrying about provisioning and scaling servers. On the other hand, LCSD mainly focuses on simplifying the application development tasks with minimum handwritten code that serverless does not address. One similarity is that the LCSD approach also aims to free the developers from the burden of deployment and maintenance tasks. LCSD tries to address the whole software development process, including quality assurance, version control, and monitoring applications. Another difference is that serverless aims to make some of the software developers' tasks easier; on the other hand, LCSD aims to include domain experts to be included in the whole SDLC phases. In summary, both approaches' goal is faster app development. Nevertheless, in practice, they have adopted quite a different approach. Recent development in enterprise applications is moving towards containers and micro-service architecture. This trend has positively impacted the LCSD approach for large organizations.

2.1.1.2 Development Phases of an LCSD Application

Software development methodology helps to select proper strategies to develop software applications. They include various methods to define the software development life cycle (SDLC). The waterfall and agile methodologies are widely used models in various software projects. Software is developed with specific requirements and goes through several phases, such as analysis, coding, and testing. Software engineering is a development process for software products to ensure quality, cost, and meeting deadlines. LCSD team employ any software development methodologies [217], but many of the LCSD platforms (e.g., Oracle Apex [18], Appian [21] etc.) provide in-build support for the following SDLC methodologies in their platform: (1) *Agile software development [48]* is an iterative and incremental way of developing a software application where the requirements and the functionalities of the system evolve. Besides the software industry, many other industries, such as aerospace, banking, and construction, have adopted this methodology. (2) *Rapid application development (RAD) [49]* is a software development methodology that promotes the rapid release of a software prototype. It is an agile approach and aims to take user feedback from the prototype into consideration to deliver a better product. (3) *Iterative software development [45]* is based on the iterative development of the application. In this way, every step is cyclically repeated one after another. In practice, this is very helpful because it allows for the development and improves the application gradually.

A typical LCSD application can be built in two ways [217]: 1. “UI to Data Design”, where developers create UI and then connect the UI to necessary data sources, or 2. “Data to UI”, where the design of the data model is followed by the design of the user interfaces. In both approaches, application logic is implemented, and then third-party services and APIs are integrated. APIs are interfaces to reusable software libraries [212]. A major motivation behind LCSD is to build applications, get reviews from the users, and incorporate those changes quickly [263]. Some software development approaches are quite popular and supported by different LCSD platforms, such as Iterative software development [45], which is based on the iterative development of the application. In this way, every step is cyclically repeated one after another. In practice, this is very helpful because it allows the development and improving the application gradually. Another approach can be Rapid application development (RAD) [49] is a software development methodology that promotes the rapid release of a software prototype. It is an agile approach that utilizes user feedback from the prototype to deliver a better product. Another popular methodology is the agile development methodology [48] which is a collection of approaches and practices that promote the evolution of software development through collaboration among cross-functional teams.

Different LCSD teams may adopt different SDLC approaches. However, we focus mostly on Agile methodology for this study because Agile and LCSD can go hand in hand. After all, the fundamental principle and objectives are customer satisfaction and continuous incremental delivery. Traditional software development teams widely use agile, providing generalizability for other methodologies. So, in this study, we map agile software development life cycle phases with LCSD methodologies. The inner circle of Figure 3.1 shows the important development phases of an LCSD application, as outlined in [217]. The outer circle of Figure 3.1 shows the phases in a traditional agile software development environment. As LCSD platforms take care of many application development challenges, some agile application development phases have shorter time/execution spans in LCSD than traditional software development.

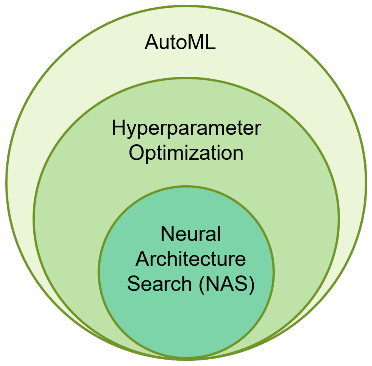


Figure (2.6) An overview of AutoML for discovering the most effective model through Neural Architecture Search (NAS) [31]

2.1.2 Low-code in machine learning development

Recent advancements in machine learning (ML) learning have yielded highly promising results for various tasks, including regression, classification, clustering, etc., on diverse dataset types (e.g., texts, images, structured/unstructured data). The development of an ML model requires significant human expertise. Finding the optimal ML algorithm/architecture for each dataset necessitates intuition based on experience. ML-expert and domain-expert collaboration is required for these laborious and arduous tasks. The shortage of ML engineers and the tedious nature of experimentation with different configuration values sparked the idea of a low-code approach for ML. This low-code machine learning solution seeks to solve this issue by automating some ML pipeline processes and offering a higher level of abstraction over the complexities of ML hyperparameter tuning, allowing domain experts to design ML applications without extensive ML expertise. It significantly increases productivity for machine learning practitioners, researchers, and data scientists. The primary goal of low-code AutoML tools is to reduce the manual efforts of different ML pipelines and thus accelerate their development and deployment.

2.1.2.1 AutoML

In general terms, a machine learning program is a program that can learn from experience, i.e., data. In the traditional approach, a human expert analyses data and explores the search space to find the best model. AutoML [73,126] aims to democratize machine learning to domain experts by automating and abstracting machine learning-related complexities. It aims to solve the challenge of automating the Combined Algorithm Selection and Hyper-parameter tuning (CASH) problem. AutoML is a combination of automation and ML. It automates various tasks on the ML pipeline, such as data prepossessing, model selection, hyperparameter tuning, and model parameter optimization. They employ various techniques, such as grid search, genetics, and Bayesian algorithms. Some AutoML services help with data visualization, model interpretability, and deployment. It helps non-ML experts develop ML applications and provides opportunities for them to engage in other tasks [109]. The lack of ML experts and exponential growth in computational

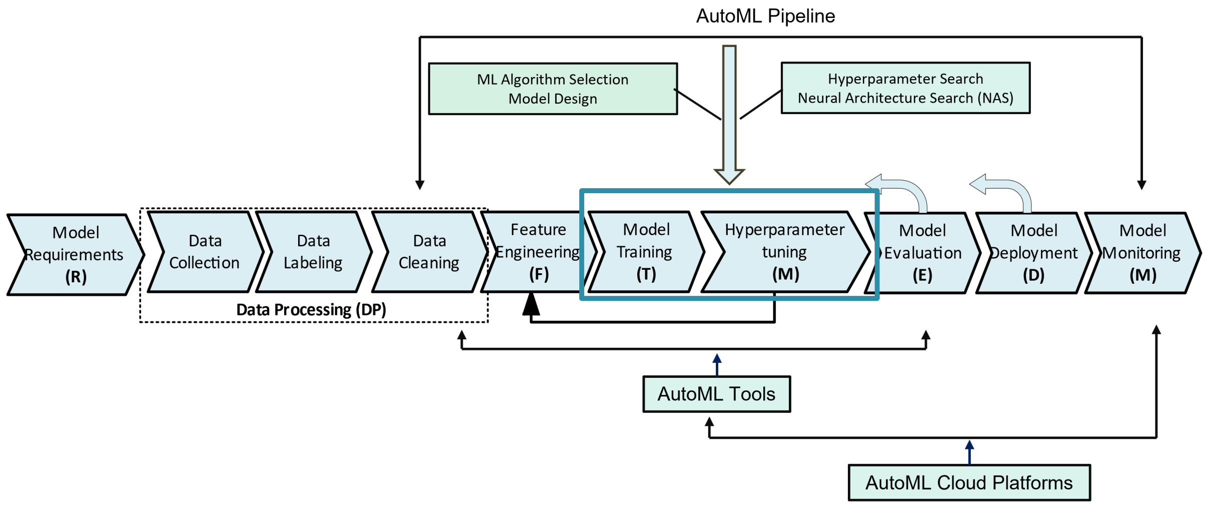


Figure (2.7) An overview of traditional ML pipeline vs. AutoML tools/platform pipeline.

power make AutoML a hot topic for academia and the industry. Research on AutoML research is progressing rapidly; in some cases, at least in the experimental settings, AutoML tools are producing the best hand-designed models by ML experts [167].

AutoML approach can be classified into two categories (1) AutoML for traditional machine learning algorithms. It focuses on data pre-processing, feature engineering (i.e., finding the best set of variables and data encoding technique for the input dataset), ML algorithm section, and hyperparameter tuning. (2) AutoML for deep learning algorithms: this includes Neural architecture search (NAS), which generates and assesses a large number of neural architectures to find the most fitting one by leveraging reinforcement learning [233] and genetic algorithm [176]. Figure 4.1 provides a high-level overview of AutoML for NAS and hyper-parameter optimization. The innermost circle represents the NAS exploration for the DL models, and the middle circle represents the hyper-parameter optimization search both for NAS and traditional ML applications.

AutoML Tools. The initial AutoML tools were developed in partnership with academic researchers and later by startups and large technology corporations [242]. Researchers from the University of British Columbia and Freiburg Auto-Weka (2013) [146], one of the first AutoML tools. Later, Researchers from the University of Pennsylvania developed TPOT (2014) [184], and researchers from the University of Freiburg released Auto-sklearn (2014) [100]. These three AutoML tools provide a higher level abstraction over the popular ML library "SciKit-Learn" (2007) [195]. A similar research effort was followed to provide an automated ML pipeline over other popular ML libraries. University of Texas A&M University developed Auto-Keras (2017) [136] that provides runs on top of Keras [117] and TensorFlow [223]. Some of the other notable AutoML tools are MLJar (2018) [200], DataRobot (2015) [74], tool named "auto_ml" (2016) [28]. These AutoML tools provide a higher level of abstraction over traditional ML libraries such as Tensorflow, Keras, and Scikit-learn and essentially automate some ML pipeline steps (i.e., algorithm selection

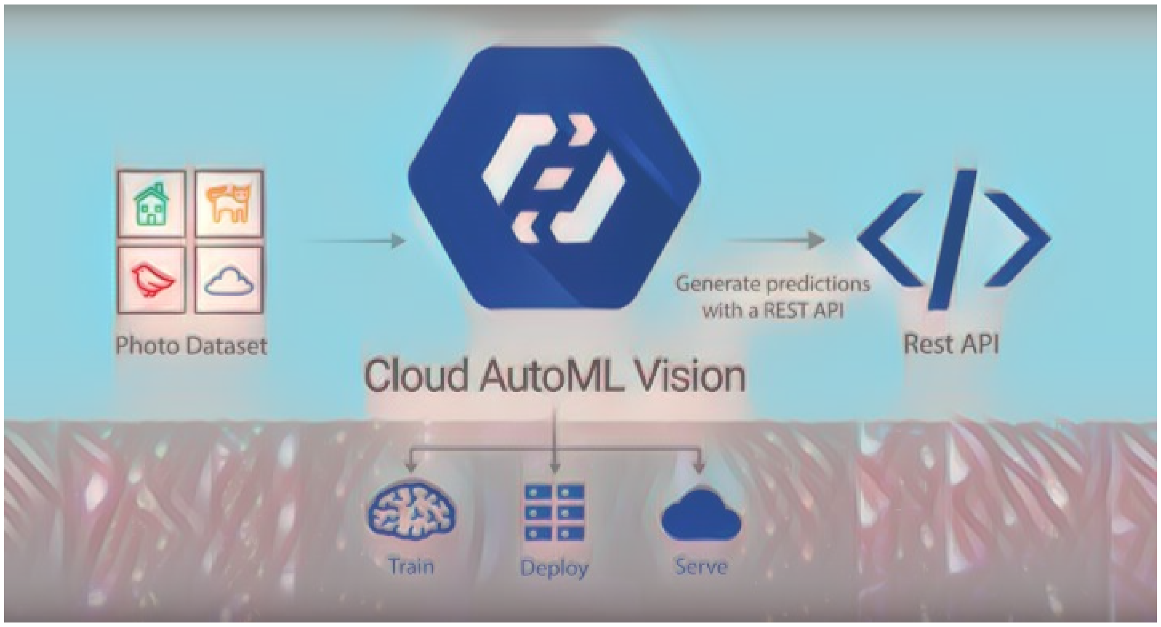


Figure (2.8) A high-level overview of Google Cloud AutoML solution [1]

and hyper-parameter turning).

AutoML Platforms. Large cloud providers and tech businesses started offering Machine Learning as a Service projects to make ML even more accessible to practitioners due to the rising popularity of AutoML tools. Some of these platforms specialize in various facets of AutoML, such as structured/unstructured data analysis, computer vision, natural language processing, and time series forecasting. In 2016, Microsoft released AzureML [234] runs on top of Azure cloud and assists ML researchers/engineers with data processing and model development. H2O Automl [150] was released in 2016, followed by H2O-DriverlessAI [120] in 2017. It is a customizable data science platform with features such as automatic feature engineering, model validation and selection, model deployment, and interpretability. In 2017 Google released Google Cloud AutoML [115] that provides end-to-end support to train the custom models on custom datasets with minimal effort. Some other notable cloud platforms are Darwin (2018) [72] AutoML cloud platform for data science and business analytics, TransmogrifAI (2018) [240] runs on top of Salesforce's Apache Spark ML for structured data. This cloud-based AutoML platform enables end-to-end data analytics and AI solutions for nearly any sector.

In Figure 4.2, we summarize the ML pipeline services offered by AutoML tools/platforms. Traditional ML pipeline consists of various steps such as Model requirement, Data processing, feature engineering, model deigning, model evaluation, deployment, and monitoring [9, 15]. AutoML tools/platforms aim to automate various stages of these pipelines, from data cleaning to Model deployment [242] (Fig. 4.2). AutoML tools mainly automate different data filtering, model selection, and hyperparameter optimization. AutoML cloud platforms encapsulate the services of AutoML tools and provide model deployment and monitoring support. They usually provide the necessary tools for data exploration and visualization.

Figure 2.8 provides a high-level overview of the Google Cloud AutoML platform. It provides a visual interface where domain experts can upload images or textual data. After that, the domain expert can process the dataset and train an ML application with a few clicks. They can even deploy the application via REST API with a few mouse clicks. They usually provide the necessary tools for data exploration, visualization, and model interpretability.

Replication Package is available in `https://github.com/disa-lab/Thesis-Alamin-MSc/tree/main/Chapter_2`

2.2 Characterization of Traditional Low-code platforms

2.2.1 Motivation

With the rising demand for automation, the LCSD methodology is gaining popularity among businesses. There are hundreds of recognized tech companies that provide low-code solutions for a variety of applications. With the rising shortage of skilled software engineers, the demand for the low-code solution is expected to grow over time, and by 2024 around 65% of businesses are expected to adopt this methodology to some extend [266]. In this section, we undertake a qualitative evaluation of existing low-code platforms. We wish to investigate their support for different types of application development (e.g., workflow automation, web form processing, etc.), the distribution of open-source vs. proprietary LCSD platforms, and the status cloud deployment option (i.e., vendor lock-in), target application platform (e.g., mobile, web or desktop). This analysis will provide valuable insight into the current state of low-code application development technologies. This analysis will provide researchers, software developers, low-code providers, and the OSS community with useful insight into the current state and opportunities for future development.

2.2.2 Approach

We gather and analyze traditional low-code platforms in two steps: (1) Collect a list of traditional LCSD vendors, (2) Categorize each traditional LCSD service, and We describe the steps below.

Step 1: Collect a list of LCSD vendors. we compile a list of top LCSD vendors by analyzing a list of LCSD services that are considered as the market leaders in Gartner [258], Forrester [216], related research work [217], and other online resources like PC magazine [194]. We explore all these resources and compile a list of 121 LCSD service providers.

Step 2: Manually categorize the services. Then we manually categorize each LCSD service based on its web page description. We categorize the selected LCSD platforms from different aspects:(1) Types of supported application development, (2) Software licensing, (3) Deployment flexibility, (4) Target platform.

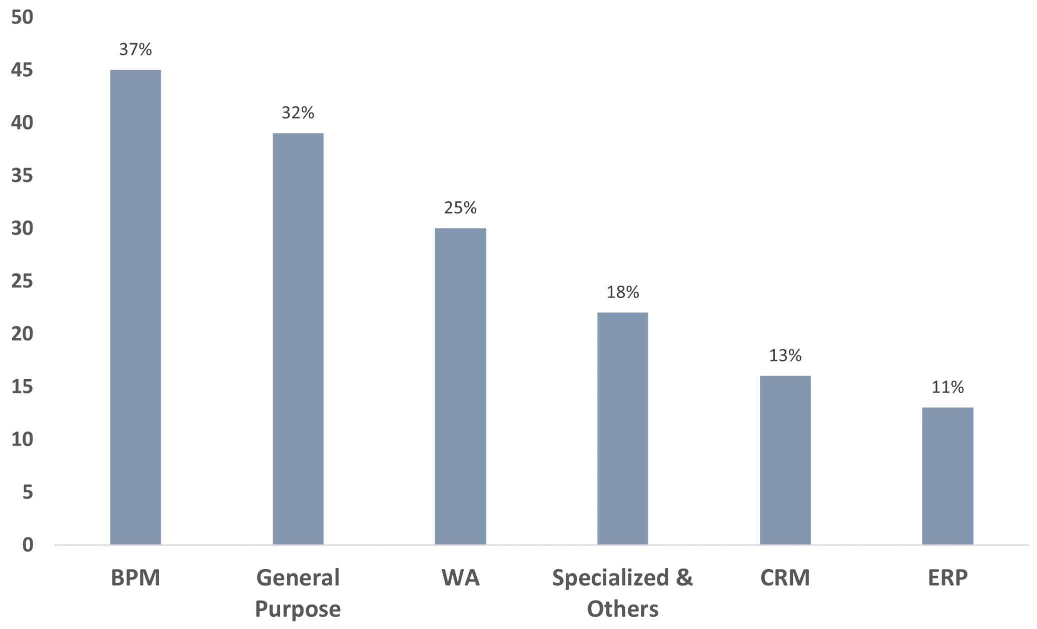


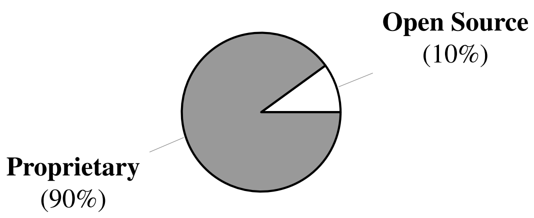
Figure (2.9) A distribution of different types of applications supported by LCSD platforms.

• **Application Type.** LCSD platforms provide a wide range of application development options. Some platforms offer a versatile development environment for any general-purpose application, while others are designed to address specific business problems, such as workflow automation and customer relationship management. We assign one or more of these application types to each LCSD platform. (1) Customer relationship management (CRM) [196], (2) Business Process Management (BPM) [123], (3) Workflow Automation (WA), (4) Enterprise Resource Planning (ERP), (5) General purpose applications, (6) Specialized & Other applications

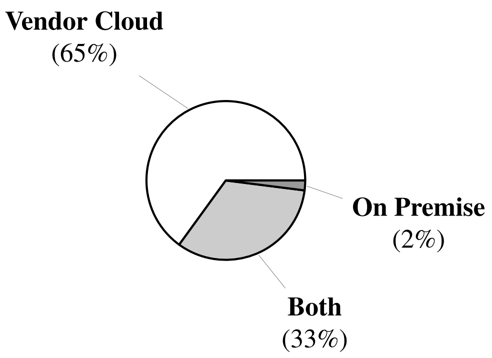
• **Software licensing.** We classify each LCSD platform as either open-source software (OSS) or proprietary software (proprietary code): (1) Open Source: if the source code of the platform is publicly available for change and modification, (2) Proprietary: if the LCSD platform is developed and maintained by a person, team, or organization who maintains exclusive control over the platform.

• **Deployment flexibility.** We categorize each LCSD platform into one of these three options: (1) Vendor Cloud: if the developed application can only be deployed and monitored via the vendor's cloud solution, (2) On-Premise: LCSD platforms offer a development environment but not deployment options, (3) Both: if the LCSD platform offers proprietary cloud services for easy application release and an on-premise deployment option.

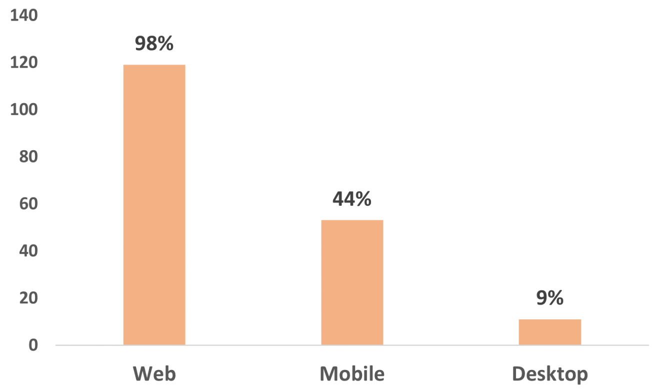
• **Target platform.** We classify each LCSD platform based on the availability of application development options for these three target platforms: (1) Web, (2) Mobile, (3) Desktop.



(a) Open Source vs. Proprietary LCSD platforms.



(b) Distribution of deployment flexibility for traditional LCSD platforms.



(c) Distribution of LCSD platforms on the target application domain.

2.2.3 Result

We find that all of our analyzed LCSD services (i.e., 121) fall under the platform category (Fig. 2.1 in Section 5.3). These LCSD platforms empower citizen developers or domain experts to contribute to the development process actively. This rapid development and deployment enable businesses to react faster to the ever-changing market demand. We provide an overview of LCSD platforms for traditional application development below.

Application Type. LCSD platforms offer a wide range of features for different use cases for stakeholders. In Figure 2.9, we present the distribution of different types of applications by LCSD platforms. An LCSD platform can offer support for multiple types of application development. Most LCSD platforms offer support for business processes such as form, workflow management, and general-purpose applications. We provide a detailed description below.

- *Business Process Management (BPM) [123] (37%):* is the biggest type of application that most LCSD platforms offer. A business process is a discipline to optimize where an organization analyzes its processes and tries to improve the organization. It coordinates different stakeholders and systems to accomplish some business goal. It can be structured or unstructured. Some of the LCSD platforms provide necessary tools so that different departments in the organization, such as HR, finance, and Sales, can automate many tasks, such as digitizing the approval process of buying new equipment, travel requests, hiring new employees, etc.

- *General purpose applications (32%):* is the second largest type are applications. These applications are developed for a wide range of business or personal purposes. They can be as simple as a website, a simple form-based data collection application, a desktop/web application for data processing, etc., for developing digital applications better and faster. LCSD platforms provide a wide range of tools to easily design UI, implement business logic and deploy the application in the managed cloud or on-premise servers.

- *Workflow Automation (WA) (25%):* is the process that helps to automate and steam line a series of works that is required for business operation. It can empower employees by eliminating unnecessary manual tasks. It involves automating business-critical tasks, organizing, storing documents, and sharing across multiple people and systems based on predefined business rules. It can improve productivity and job satisfaction. LCSD platforms provide drag and drop interface with pre-built tasks that can be customized. It provides tools for various tasks, such as processing forms and notifying stakeholders. BPM and workflow automation tools have similarities, but they have certain differences. Workflow automation helps manage repetitive tasks by automating certain procedures. BPM is a more complex approach; it not only aims to automate a business process but rather guidelines to improve each process continuously.

- *Specialized & Others (18%):* is the LCSD platform that helps to develop specialized applications such as Robotic Process Management (RPA), IoT, game development(e.g., Struct [229]), cross-platform mobile development (e.g., Xojo [272]), etc. RPA is a technology that provides the necessary tools to build and manage software or physical robots that can interact with other systems. It tries to mimic human tasks, such as interacting with computer screens and navigation systems, triggering responses. LCSD platforms provide necessary tools that can perform a wide range of tasks, from automated email replies to deploying bots. They are usually a specialized part of ERP systems.

- *Customer relationship management (CRM) [196] (13%):* is a process of managing the relationship of a business or organization with its existing customers or potential customers. The goal is to improve the relationship and grow the business. A CRM system can streamline the process of customer management by providing tools to manage contacts, sales information, and promotional offer. LCSD platforms provide the tools to analyze business data and insights about the target audience and improve the relationship by providing necessary support and services over time.

- *Enterprise Resource Planning (ERP) [255] (11%):* consists of an integrated suite of business-related applications. ERP tools cover a wide range of end-to-end processes from finance to HR to manufacturing to supply chain. It is a complex endeavor that coordinates processes from multiple industries. Some LCSD platforms provide various ERP tools to help businesses manage their regular business activities, such as project management, HR, accounting, and compliance. Large LCSD platforms such as Mendix [171] and Zoho Creator [283] offer ERP complete low-code ERP solutions for medium to large businesses.

Software Licensing. It usually determines the condition and cost of using an LCSD platform. As most of the LCSD platforms target commercial businesses, the majority of them are proprietary. Fig. 2.10a provides an overview of the proprietary (90%) vs. open-source (10%) LCSD platform.

- *Open Source (10%):* Generally, all OSS LCSD platforms are free to use, and enterprise-grade features are available for a fee. These platforms offer more flexibility in terms of deployment. For example, from our analyzed open-source LCSD platforms, 92% offer their cloud servers for a fee.
- *Proprietary (90%):* Most of the LCSD platforms are proprietary. These platforms provide a greater number of features but with less flexibility. Some of these platforms (i.e., Appian) may require some proprietary scripting language for customization. Around 69% of the proprietary platform only offers proprietary deployment solutions (i.e., the developed application will be completely locked-in to that particular vendor.)

Deployment Flexibility refers to the flexibility to deploy and run the applications on on-premise servers of any other cloud platforms. It is easier for LCSD to deploy, monitor, and scale the applications on their proprietary servers. However, this vendor lock hinders business for low-code adoption. From Fig. 2.10b, we can see that around 98% LCSD offers their vendor cloud platforms for app deployment, and approximately 33% LCSD platforms also offer support for on-premise or third-party cloud deployment. Open-source LCSD platforms do not have any vendor lock-ins.

- Vendor Cloud (65%). Most of our analyzed LCSD platforms offer support for continuous delivery, on-demand scaling, and security on their manager cloud. This provides a simple deployment solution at the expense of becoming dependent on the vendor's cloud infrastructure.
- On-Premise (2%). We find that only a few (e.g., 2%) platforms offer only on-premise application deployment. These low-code solutions provide IDE to develop and customize mobile visually (e.g., Xojo [272]) or desktop applications (e.g., Lianja App builder [157]). This strategy is not often adopted for web applications since it makes deployment difficult, especially for citizen developers.
- Both (33%). This is the best of both worlds. We find that around 33% LCSD platforms offer cloud services for easy application release and on-premise deployment to help businesses avoid vendor lock and comply with customer data privacy policy.

Target Platform. Most of the LCSD platforms (i.e., around 98%) offer to develop web-based cross-platform applications that can be accessed from Desktop and mobile via a web browser. However, we also find that some platforms such as Xojo [272], Appery.io [20] offer to develop cross-platform mobile applications. In Figure 2.10c, we present the distribution of our analyzed LCSD platform based on the target platform. We find that 33% LCSD platforms offer both web and mobile application development.

- *Web (98%).* We find LCSD platforms are predominantly used for dynamic web-based applications without an in-depth understanding of web technologies (e.g., HTML, JavaScript, CSS, etc.).
- *Mobile (44%).* We also find that many LCSD platforms support native cross-platform (e.g., Android, IOS) mobile application development in addition to web-based applications.
- *Desktop (8%).* We also find that around 8% of platforms offer to develop desktop applications.

Characterization of Traditional Low-code platforms. We conduct a qualitative analysis of different aspects of 121 traditional LCSD services, which all fall under the platform category. We discover that 37% of LCSD platforms offer services for developing business process management (BPM)-related applications, followed by general-purpose apps (32%) and work process automation (25%) related applications. Only about 10% of typical LCSD platforms are open-source, while the remaining 90% are proprietary. 63% of LCSD platforms offer a proprietary cloud deployment option (i.e., vendor lock-in) solely, 34% of platforms offer both a vendor cloud deployment option and a third-party deployment option, and only 2% of LCSD platforms do not offer any cloud services. Approximately 98% of LCSD platforms exclusively support web-based application development, followed by mobile (44%), Desktop, and other specialized applications (9%), and 33% platforms support both web and mobile.

2.3 Characterization of Low-code ML tools/platforms

2.3.1 Motivation

Low-code approach for ML is gaining increasing attention from academia, industry, and domain expert practitioners. More and more businesses are adopting data-driven decision-making and are expected to reach a market share of $14 Billon by 2030 [30]. Initially, academic researchers were leading the research. However, the IT industry soon realized its potential and began offering services for domain specialists able to apply ML solutions directly based on commercial demands. We undertake a qualitative evaluation of existing AutoML platforms and tools. We intend to study the AutoML-based features offered by existing service providers, the current status of collaboration between open-source research and proprietary AutoML solutions, and the ML application types that these services specialize in. This analysis will provide ML and SE researchers, AutoML vendors, and practitioners with valuable insights into the present state of AutoML research and the market and future potential.

2.3.2 Approach

Similar to Section 2.2, we gather and analyze AutoML tools/platforms in two steps: (1) Collect a list of AutoML tools/platforms, (2) Categorize each AutoML tool/platform, and We describe the steps below.

Step 1: Collect a list of AutoML tools/platforms. First, we compile a list of top AutoML tools/platforms. We compile a list of AutoML tools by different sources: (1) relevant studies on AutoML services (e.g., AutoML growth forecast [30], Benchmark studies on different AutoML tools [222, 242], (2) popular technological research website Gartner [108], G2 [106], (3) by performing a google query form incognito browser window with the following query term "AutoML tools/platform" to get a list of tech websites that rank the best AutoML tools/platforms. The full list of these tech websites is available in our replication package. We compile and create a list of 37 AutoML tools/platforms for this analysis from all these resources.

Step 2: Manually categorize the AutoML tools/platforms. Then we manually categorize each platform based on their respective web page or GitHub page description. We categorize the selected AutoML tools/platforms from different aspects:(1) ML Application category, (2) AutoML type (i.e., tool vs. cloud platform), (3) Software licensing, (4) Deployment flexibility

• **ML Application Category.** We classify each AutoML tool or platform as one of the four ML solution categories based on the following criterion: (1) *Shallow ML Model:* if the AutoML service offers to optimize ML pipeline and hyperparameter optimization for various traditional shallow ML algorithms such as Logistic Regression, k-Nearest Neighbours, Gaussian Naive Bayes, and Random Forests/Decision Trees, etc. (2) *Deep Learning Model:* if the AutoML tool or platform offers a high level of abstraction over deep neural networks (DNN) for DL models used in computer vision, natural language processing, or time series forecasting problems. (3) *General ML Application:* If the AutoML tool or platform provides an intelligent machine learning (ML) solution for various datasets using a high-level abstraction of shallow ML and DL algorithms for regression, classification, and other tasks. (4) *Data Analytic Application:* if the platform/tools provide different services to identify patterns and generate insights to help decision-making by utilizing different data analytics techniques on the raw data (e.g., structured, unstructured, qualitative data). These services combine intelligence with action by focusing on business datasets.

• **AutoML Type.** Based on the following definition, we categorize any AutoML tool or platform as one of two types: (1) *Tool/framework:* if the AutoML solution only offers programming API to automate different ML pipelines (e.g., AutoKeras [136], AutoSklearn [100]). In most cases, the practitioners must configure their development environment. (2) *Cloud Platform:* if the AutoML service provides easy to use pre-configured development environment via a visual

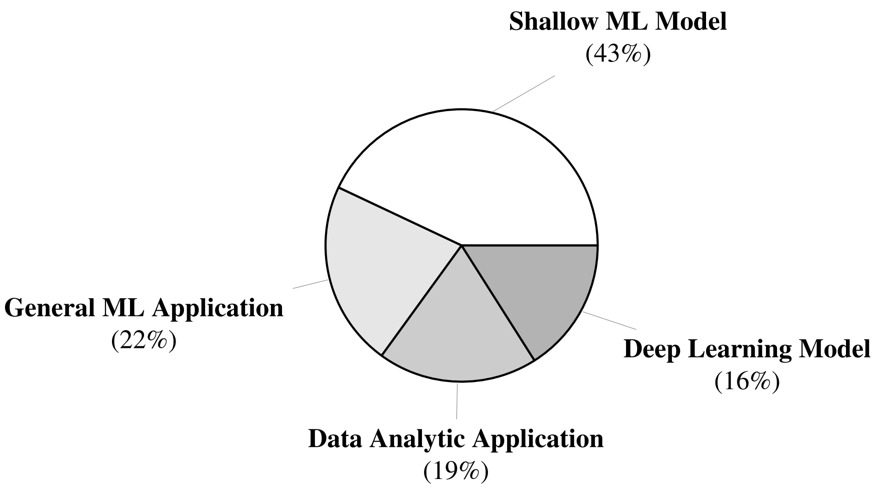


Figure (2.11) Distribution of different types of ML application development solutions offered by AutoML vendors.

interface (mostly via web platforms). They also offer easy data management, model training, and deployment options.

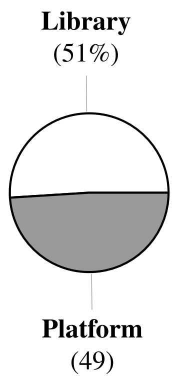
As described in Section 2.2.2, we manually classify each AutoML tool or platform as either open-source software (OSS) or proprietary software. Similarly, we categorize the AutoML services based on their deployment flexibility (e.g., Vendor Cloud, On-premise deployment, or both.) We further categorize open-source AutoML services into academic or industry development based on the following guideline:(1) *Academia:* if the tool or platform is developed or maintained by the researchers in collaboration with a university, (2) *Industry:* If the tool or platform is open-source, yet was created and is maintained by a tech company.

2.3.3 Result

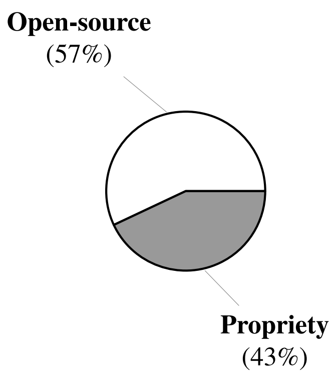
AutoML services empower data scientists and domain experts to develop, deploy, and manage effective ML models faster. They provide services for machine learning operations (MLOps), integrated services, open-source interoperability, explainability, security, and scale on demand. It enhances automatic decision-making by removing ML's complexities so businesses can focus on what matters most.

ML Application Category. AutoML tools or platforms offer various services for applying various ML algorithms to a dataset. In Figure 2.11, we illustrate the distribution of AutoML services based on their support for different types of ML applications. Approximately 43% of our analyzed AutoML suppliers offer services for shallow machine learning type model development, compared to only 16% for deep neural network models. All these vendors aim to provide a high-level abstraction over the complexity of the ML pipeline, allowing domain experts to focus on their datasets and business goals. We provide a detailed description for each of these ML solution types:

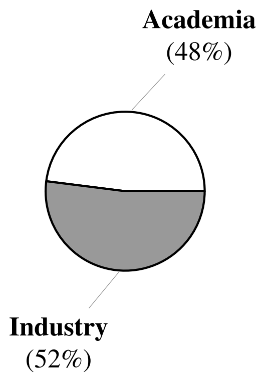
- *Shallow ML Model (43%):* is the biggest ML application category. AutoML vendors provide powerful knowledge extraction tools for feature engineering, data normalization, and other steps for regression, classification,



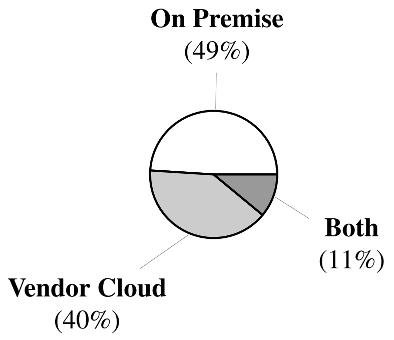
(a) Distribution of AutoML service type between library/framework and platforms.



(b) Distribution of open-source vs propriety AutoML services.



(c) Distribution of academic and industrial contributions to open-source AutoML services.



(d) Distribution of deployment flexibility for AutoML services.

or other tasks so that novice data analysts, ML users, or researchers can focus more on the insight of the data rather than writing codes. They explore classic ML algorithms with different hyperparameter configurations to find the best fit for the dataset.

- *General ML Application (22%):* is the second biggest type of ML application category. These AutoML tools/-platforms take the best from shallow ML algorithms and deep learning models to build, train and finetune the best model on the dataset. They are relatively computationally more expensive than the previous category as they explore a larger solution space.

- *Data Analytic Application (19%):* AutoML services are quite mature and target large-scale datasets from businesses. They drive business expansion by forecasting customer demands over time and predicting lifetime value for products and services. They operationalize AI by helping businesses to prioritize decisions and take action. They offer services to clean, prepare, engineer data and train hundreds of models to find the best answer. They also provide explainability (i.e., visibility and control) of their outcome to the stakeholders.

- *Deep Learning Model (16%):* is the smallest ML application category. However, DNN-based AutoML tools and platforms are gaining more and more popularity for, especially for Image (e.g., classification, segmentation), Text (e.g., summarization, sentiment detection), and time series dataset (e.g., sales forecast). These types of AutoML services are computationally expensive, but the several techniques in NAS and transfer learning is improving its performance within acceptable range [80, 130, 170, 198].

We find that the AutoML vendors offer different features to develop types of business applications in those four categories. We provide a brief description of them: (1) **Natural Language Processing (NLP) [66]** applications are from Deep Learning Model category. Businesses have many text data that they receive by various methods. AutoML aims to help build state-of-the-art NLP applications such as language translation, sentiment analysis, and text classification with minimal coding or limited exposure to machine learning (e.g., Google AutoML NLP [114]). (2) **Recommendation System [180]** applications fall under the General Application category, and they are systems that explore a huge amount of data to suggest relevant items to users. In an ever-growing technological world, this system plays a very important role in our day-to-day life whenever we use popular services such as YouTube, Netflix, and Amazon. Based on your previous choices, it suggests to us what to buy, what to watch or read. Nowadays, various businesses are trying to incorporate recommendation systems for their products or services. Many open-source frameworks and commercial cloud platforms provide easy-to-use interfaces where businesses can easily build their recommendation systems by providing only labeled data (e.g., Azure ML service [36]. (3) **Bot** development applications fall under the Deep learning model category as they utilize the latest development in NLP. The domain of user experiences is changing over time. The tech-savvy younger generation prefers solving their problem automated rather than talking to a human in a call center. Many businesses are currently employing chatbots on their website to solve customers' issues, collect feedback and even offer new products (e.g., Amazon Lex [33]). Some of the AutoML cloud providers offer the necessary tools to build a Q&A chatbot or a conversational bot. (4) **Computer Vision [189]** applications fall under the Deep Learning Model category and have many applications in e-commerce, manufacturing, and security for use cases such as image classification, object detection, face recognition, etc. AutoML aims to automate the development process of building real-world applications with target datasets (Google AutoML Vision [114]). Different AutoML tools, such as transfer learning and NAS, explore different approaches to streamline the process. In transfer learning, the neural model that already performed well for a task is used and retrained on the new tasks. It works quite well if the dataset for the new tasks is not very large. This is easy to use and computationally less expensive. On the other hand, in the NAS approach, a new neural architecture is searched that matches best for the dataset. As new neural architecture is used, transfer learning, i.e., pre-trained weights from previous models, can not be used. It usually has a control unit that trains child network architecture and finds the best child architecture that fits the dataset. (5) **Business Intelligence** applications fall under Data Analytic Application or Shallow ML Model category, and it refers to the process where various business insights help the stakeholder to make data-driven decisions by discovering trends and probable outcomes (e.g., RapidMiner Platform [207]). It consists of methodologies such as data mining, data aggregation, statistical analysis, forecasting, data visualization, and predictive analysis.

AutoML Type. Initially, AutoML services were designed as a high-level ML framework or library. Nonetheless, setting up the development and deployment environment needs a high level of technical expertise. With the success of traditional LCSD platforms, many large technology companies (e.g., Google AutoML Vision [51], Azure AutoML [234], H2O AutoML [119]) began to offer an end-to-end AutoML cloud platform to support quicker data processing from other cloud platforms and to provide high-end computational resources on demand for model training and deployment. In Fig. 2.12a, we depict the current condition of AutoML tools vs. platforms and demonstrate that, according to our analyzed dataset, AutoML tools (51%) continue to outnumber AutoML platforms (49%).

- *AutoML tool (library/framework) (51%).* Still, most of the AutoML services are provided as a tool, i.e., a library usually without UI. We find that all the AutoML tools (e.g., 100%) are open-source and dominantly (e.g., 53%) from academia (e.g., AutoWeka [146], AutoGluon [81], Auto-SKlearn [100]).

- *AutoML cloud platform (49%).* AutoML platforms provide end-to-end support for data processing, model training, and deployment support. Many low-code services provide intelligent data analytics services (e.g., Rapid Miner [207]) to benefit from the business data. There still requires some expertise to design high-performing ML models, especially for the DNN models. However, recent advancement in AutoML research is a promising outcome, and the cloud-based AutoML services are expected to grow rapidly for their flexibility and scalability [30].

Software Licensing. determines the legal condition for developing and training an ML model using an AutoML service. In Figure 2.12b, we see that around 57% of our analyzed AutoML services are open-source, and the rest are from tech companies.

- *Open-Source (57%).* We find that open-source development methodology dominates in AutoML tools, i.e., all of our studied AutoML libraries/frameworks are open-source.

- *Proprietary (43%):* We find that around 90% of AutoML platforms are proprietary. These platforms offer more features for businesses, such as advanced security & monitoring, model visibility dashboard, large-scale data integration, and on-demand scalability.

From Figure 2.12b and Figure 2.12c) Overall, 57% AutoML tools are open-source, and among these open-source services, 52% are from the tech industry, and 48% are from academia. This highlights the strong collaboration between academia and industry and their contribution and commitment to open-source software development for this emerging research domain.

Deployment Flexibility. refers to the control and flexibility to deploy the trained model on-premise servers or any cloud providers. As open-source development is more prominent in AutoML, we find that most of the services (i.e., 60%) offer deployment flexibility (Fig. 2.12d).

- *On Premise (49%):* We find that around 49% AutoML services, predominately AutoML tools offer flexible on-premise or any other cloud deployment option.
- *Vendor Cloud (40%):* We find that around 40% AutoML services, predominately AutoML platforms, offer a one-click proprietary cloud deployment option. In these cases, the developed model can not be exported to other cloud providers such as (Amazon Lex [33], Rapid Miner [207], Amazon SageMaker [35]).
- *Both (11%).* We find that around 11% of AutoML services provide proprietary clouds and on-premise or any other cloud deployment option. In this case, the model can be trained on one cloud platform but deployed anywhere (e.g., H2O AutoML [119], MLBox [177]).

Characterization of Low-code ML tools or platforms. We perform a qualitative analysis of 37 AutoML vendors. Among these vendors, 43% offer Shallow ML, 22% general ML, 19% data analytics, and 16% deep learning application development services, respectively. 51% of overall AutoML services are tools (i.e., libraries/frameworks) primarily from academia (57%), whereas the remaining 49% AutoML services are cloud-based AutoML platforms from the tech industry. We find that 57% of AutoML tools are open-source and that among these open-source services, 52% are from the tech industry and 48% are from academia, highlighting the substantial participation of academic and industrial researchers. We found that 49% of AutoML providers only offer on-premise deployment options, 40% only offer cloud-based deployment options, and only 11% are adaptable for all deployment options.

2.4 Summary

In this chapter, we discuss the motivation, key users, technologies and objectives of the LCSD approach. We provide an in-depth analysis of low-code SDLC in the context of traditional low-code and ML applications. We describe how the low-code approach is trying to automate different aspects of SDLC and ML workflows by providing better abstraction. From our qualitative assessment of 137 traditional LCSD platforms, we find that around 90% of them are propriety, with 65% of them offering only vendor-specific cloud deployment options. Around 98% of the developed applications are web-based. Similarly, from our qualitative assessment of 37 AutoML services, we find that 43% offer shallow ML application development and only 16% deep learning applications. We find that, unlike traditional LCSD services, around 51% of our studied AutoML services were non-cloud-based tools. Around 48% of our studied AutoML tools come from Academia, and around 57% of these tools are open-sourced both from academia and industry. Around 60% of AutoML services offer flexible deployment options.

Chapter 3

Challenges and Barriers of Using Low Code Software for Traditional Application Development

From the background study of Chapter 2, we see the overall low-code software development are divided into two parts: traditional software application and machine learning application. The purpose of this Chapter is to conduct an empirical study of the challenges of traditional LCSD practitioners. Software engineers frequently use the online developer forum Stack Overflow (SO) to seek assistance with technical issues. We observe a growing body of traditional LCSD-related posts in SO. This study presents an empirical study of around 33K SO posts (questions + accepted answers) containing discussions of 38 popular LCSD platforms. We use Topic Modeling to determine the topics discussed in those posts. Additionally, we examine how these topics are spread across the various phases of the agile software development life cycle (SDLC) and which part of LCSD is the most popular and challenging. Our study offers several interesting findings. First, we find 40 LCSD topics that we group into five categories: Application Customization, Database and File Management, Platform Adoption, Platform Maintenance, and Third-party API Integration. Second, while the Application Customization (30%) and Data Storage (25%) topic categories are the most common, inquiries relating to several other categories (e.g., the Platform Adoption topic category) have gained considerable attention in recent years. Third, all topic categories are evolving rapidly, especially during the Covid-19 pandemic. Fourth, the Agile Maintenance and Deployment phase is the most challenging among practitioners. The findings of this study have implications for all three LCSD stakeholders: LCSD platform vendors, LCSD developers/practitioners, Researchers, and Educators.

3.1 Introduction

There is a massive shortage of skilled software developers in this age of digitalization. According to Gartner, the demand for IT professionals will be multiple times more than supply [238, 263]. To make matters worse, training and hiring new software developers are very expensive in this rapidly evolving world. LCSD aims to address this issue by democratizing software development to domain experts and accelerating the development and deployment process. It tries to bridge the gap between the system requirement and the developer constraints, which is a common reason for long development times in complex business applications.

LCSD is a novel paradigm for developing software applications with minimal hand-coding through visual programming, a graphical user interface, and model-driven design. LCSD embodies End User Software Programming [191] by democratizing application development to software practitioners from diverse backgrounds [77]. It combines various approaches such as visual modeling, rapid app development, model-driven development, cloud computing, and automatic code generation. Low-code development tools enable the development of production-ready apps with less coding by facilitating automatic code generation. Additionally, LCSD platforms also provide more flexibility and agility, faster development time that allows responding quickly to market needs, less bug fixing, less deployment effort, and easier maintenance [77, 217]. These platforms are used to develop high-performance database-driven mobile and online applications for various purposes. As a result, low-code development is rapidly growing in popularity. According to Forrester, the LCSD platform market is estimated to reach \$21 billion by 2022. By 2024, over 65% of big companies will utilize LCSD systems to some extent, according to a Gartner report [266].

To date, there are more than 400 LCSD platforms [239], offered by almost all major companies like Google [113] and Salesforce [218]. Naturally, LCSD has some unique challenges [217]. Wrong choice of LCSD application/-platforms may cause a waste of time and resources. There is also concern about the security/scalability of LCSD applications [145]. With interests in LCSD growing, we observe discussions about LCSD platforms are becoming prevalent in online developer forums like Stack Overflow (SO). SO is a large online technical Q&A site with around 120 million posts and 12 million registered users [188]. Several research has been conducted to analyze SO posts (e.g., IoT [254], big data [37], blockchain [260] concurrency [7], , microservices [41]). The studies, however, did not analyze discussions about LCSD platforms in SO.

In 2021, we conducted an empirical study [10] by analyzing 4,785 posts (3,597 questions + 1,118 accepted answers) from SO that contained discussion about nine LCSD platforms. The study offered, for the first time, an overview of the challenges software developers face while using LCSD platforms. However, to date, there are over 400 LCSD platforms and we observed discussions about many of those platforms in SO. Therefore, it was important that we revisit our empirical study with a larger dataset of discussions about LCSD platforms in SO. In addition, given that the previous empirical study was a conference paper, the analysis was not as in-depth as we could have provided due to space limitations. Therefore, a larger-scale empirical study of the challenges developers face to adopt and use the LCSD platforms was warranted. Such insights can complement our previous empirical study [10] as well as the existing LCSD literature – which so far has mainly used surveys or controlled studies to understand the needs of low-code practitioners [12, 105, 145, 147].

Specifically, in this chapter, we present an empirical study of 33.7K SO posts relating to the top 38 LCSD platforms (according to Gartner [107]) at the time of our analysis to ascertain the interest and challenges of LCSD practitioners. We answer five research questions by analyzing the dataset.

RQ1. What topics do LCSD practitioners discuss? Given that LCSD is a novel paradigm, it is vital to study the types of topics discussed by LCSD practitioners on a technical Q&A platform such as SO. As a result, we use the topic modelling method LDA [52] on our 33.7K post dataset. We find a total of 40 LCSD topics grouped into five categories: Application Customization (30% Questions, 11 Topics), Data Storage (25% Questions, 9 Topics), Platform Adoption (20% Questions, 9 Topics), Platform Maintenance (14% Questions, 6 Topics), and Third-Party Integration (12% Questions, 5 Topics). Around 34% of questions are particular to the many supported capabilities of LCSD platforms, while the remaining 66% are regarding development activities, namely application customization. This is because the LCSD platform's features are naturally oriented around a graphical user interface (GUI) in a drag-and-drop environment. As a result, any customization of such features that are not native to the LCSD platforms becomes difficult.

RQ2. How do the LCSD topics evolve over time? We elaborate on our findings from RQ1 by examining how the observed LCSD topics evolved in SO over time. We conduct an in-depth analysis of LCSD-related discussions from 2008 to mid-2021 in SO. We discover that since 2012, discussion about LCSD has piqued community interest, which has increased significantly throughout the pandemic, i.e., since 2020. In recent years, Platform Adoption-related discussions have acquired more traction than original application customization or database query-related discussions. Future research and LCSD platform vendors should support emerging topics such as Library Dependency Management, External Web Request Processing, Platform Infrastructure API, and Data Migration.

RQ3. What types of questions are asked across the observed topic categories? From RQ1, we find some of the unique challenges for LCSD practitioners regarding Customization, Data Storage on the completely managed cloud platforms. This motivates us to explore further to understand more of those challenges. For instance, we want to understand if practitioners mostly ask about different solution approaches (i.e., How-type) or further explanation clarification type (Why/What-type). Following previous studies [2, 254], we manually annotated a statistically significant number of posts (e.g., 471 Questions) into four categories. We find that How-type (57%) is the most common form of inquiry across all five topic categories, followed by What-type (18%), Why-type (14%), and Other-type (12%) questions. Most of the How-type questions are application implementation-related, and most of the What-type and

Why-type Questions are server configuration and troubleshooting related. According to our findings, proper documentation and tutorials might significantly reduce these challenges.

RQ4. How are the observed topic categories discussed across SDLC phases? Our findings from the previous research questions examined the practitioners' challenges on LCSD platforms and their evolution. The acceptance of this emerging technology depends largely on effective adoption into the various stages of a software development life cycle (SDLC). So, following our previous study [10] we manually annotate statistically significant samples (e.g., 471 Questions) into six agile SDLC stages. We find that the Implementation (65%) is the most prominent phase in terms of the number of questions, followed by Application Design (17%) and Requirement Analysis & Planning (9.1%).

RQ5.What LCSD topics are the most difficult to answer? LCSD practitioners face many different challenges to understand different features of the cloud platform, server configuration. LCSD vendors aim to provide support from requirement gathering to deployment and maintenance, but practitioners still struggle with customization, data management, and cloud configuration. We find that, while the topic of application customization and the Implementation-SDLC are the most prevalent, Platform Adoption topic category and the Deployment-SDLC and Maintenance-SDLC as the most popular and hardest to get accepted answers.

Our study findings can enhance our understanding of the developers' struggle while using LCSD platforms. The findings would help the research community and platform vendors better focus on the specific LCSD areas. The practitioners can prepare for difficult areas. LCSD platforms can design more effective and usable tools. All stakeholders can collaborate to provide enhanced documentation assistance. The LCSD vendors can support increased customization of the LCSD middleware and UI to make the provided functionalities more usable.

Replication Package: The code and the datasets generated during this study is available in our public GitHub repository `https://github.com/disa-lab/Thesis-Alamin-MSc/tree/main/Chapter_3`

3.2 Background

This section aims to provide a high-level overview of LCSD development, as well as some of the relevant technologies and research that have shaped this industry. We hope that this will serve as a resource for future researchers (particularly those interested in the underlying technologies)) and practitioners to learn and contribute more to this emerging new field.

Low-code Software Application. To cater to the demand of the competitive market, business organizations often need to quickly develop and deliver customer-facing applications. LCSD platform allows the quick translation of the business requirement into a usable software application. It also enables citizen developers of varying levels of software development experience to develop applications using visual tools to design the user interface in a drag-

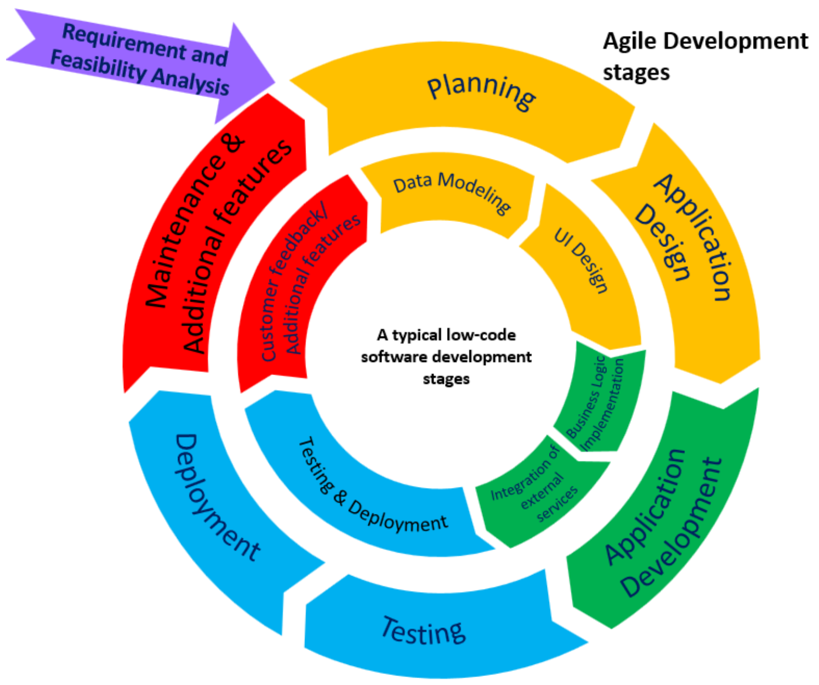

Requirement and Feasibility Analysis
Agile Development stages
Planning
Application Design
Application Development
Testing
Deployment
Maintenance & Additional features
Data Modeling
UI Design
Business Logic Implementation
Integration of external services
Testing & Deployment
Customer feedback/ Additional features
A typical low-code software development stages

Figure (3.1) Agile methodologies in traditional vs LCSD development

and-drop manner and deploy them easily [164]. LCSD is inspired by the model-driven software principle where abstract representations of the knowledge and activities drive the development, rather than focusing on algorithmic computation [217]. LCSD platforms aim to abstract away the complexity of testing, deployment, and maintenance that we observe in traditional software development. Some of the most popular low-code platforms are Appian [21], Google App Maker [113], Microsoft Powerapps [203], and Salesforce Lightning [218].

Technologies that Shaped LCSD. Model-driven Software Engineering (MDSE) field proposes the adoption of domain-specific modeling practices [58]. Low-code platforms adopt model-driven engineering (MDE) principles as their core that has been applied in several engineering disciplines for the purpose of automation, analysis, user interface design [55, 57, 199] and abstraction possibilities enabled by the adoption of modelling and meta modeling [44]. Besides, End-User Development (EUD) is a set of methods, techniques, and tools that allow users of software systems, who are mainly non-professional software developers, at some point to create, modify or extend a software artifact [103, 193]. EUD for GUIs can be a good example of its usage [69]. Scratch [210], Bloqqi [104], EUD-MARS [8], App Inventor [265], AppSheet [112] are such “low-code/no-code” application development tools that offer visual drag-and-drop facilities. Similarly, there are several other research areas within the domains of HCI [224] and Software engineering, such as Visual Programming [60], Programming by example [122], End users programming [179], domain specific language [172, 257], trigger action programming [256] that aim to enhance the technologies underlying low-code software development. Thus, gaining a better knowledge of the problems associated with low-code platforms through developer discussion would be extremely beneficial for further improving these studies.

Development Phases of an LCSD Application. A typical LCSD application can be built in two ways [217]: 1. "UI to Data Design", where developers create UI and then connect the UI to necessary data sources, or 2. "Data to UI" where the design of the data model is followed by the design of the user interfaces. In both approaches, application logic is implemented, and then third-party services and APIs are integrated. APIs are interfaces to reusable software libraries [212]. A major motivation behind LCSD is to build applications, get reviews from the users, and incorporate those changes quickly [263]. Some software development approaches are quite popular and supported by different LCSD platforms, such as Iterative software development [45] which is based on the iterative development of the application. In this way, every step is cyclically repeated one after another. In practice, this is very helpful because it allows developing and improving the application gradually. Another approach can be Rapid application development (RAD) [49] is a software development methodology that promotes the rapid release of a software prototype. It is an agile approach and aims to utilize user feedback from the prototype to deliver a better product. Another popular methodology is the agile development methodology [48] which is a collection of approaches and practices that promote the evolution of software development through collaboration among cross-functional teams.

Different LCSD teams may adopt different SDLC approaches. However, we focus mostly on Agile methodology for this study because Agile and LCSD can go hand in hand because the fundamental principle and objective are customer satisfaction and continuous incremental delivery. Traditional software development teams widely use agile, which also provides the generalizability for other methodologies. So, in this study, we map agile software development life cycle phases with LCSD methodologies. The inner circle of Figure 3.1 shows the important development phases of an LCSD application, as outlined in [217]. The outer circle of Figure 3.1 shows the phases in a traditional agile software development environment. As LCSD platforms take care of many application development challenges, some of the agile application development phases have shorter time/execution spans in LCSD than traditional software development.

3.3 Study Data Collection and Topic Modeling

In this Section, we discuss our data collection process to find LCSD related posts (Section 4.3.1). We then discuss the details about our pre-processing and topic modeling steps on the selected posts (Section 4.3.2).

3.3.1 Data Collection

We collect LCSD related SO posts in three steps: (1) Download SO data dump, (2) Identify LCSD related tag list, and (3) Extract LCSD related posts from the data dump based on our selected tag list. We describe the steps below.

Step 1: Download SO data dump. For this study, we used the most popular Q&A site, Stack Overflow (SO), where

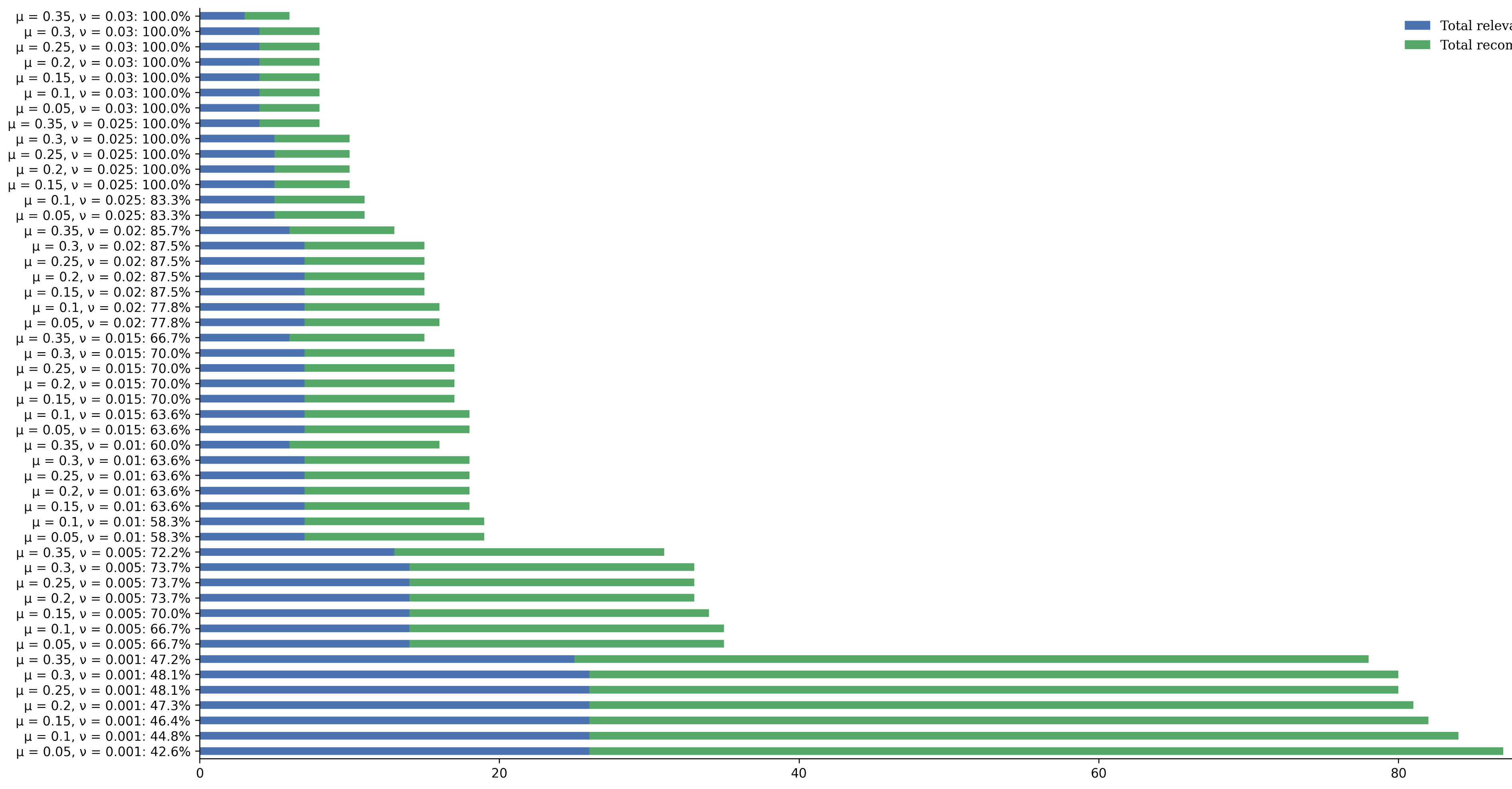


Figure (3.2) Distribution of LCSD related recommended vs relevant tags based on different *u* and *V* values from our initial taglist.

developers from diverse background discuss about various software and hardware related issues [188]. For this study, We downloaded SO data dump [97] of May 2021 which was the latest dataset available during the starting of this study. We used the contents of "Post.xml" file, which contained information about each post like the post's unique ID, type (Question or Answer), title, body, associated tags, creation date, view-count, etc. Our data dump included discussion of 12 years from July 2008 to July 2021 and contained around 53,086,327 posts. Out of them, 21,286,478 (i.e., 40.1%) are questions, 31,799,849 (i.e., 59.9%) are answers, and 51.5% questions had accepted answers. Around 12 million users from all over the world participated in the discussions.

Each SO post contains 19 attributes, and some of the relevant are: (1) Post's body with code snippets, (2) Post's Id, creation and modification time, (3) Post's view count, favorite count, score, (4) User Id of the creator, (5) Accepted answer Id and a list of 0 to 5 tags.

Step 2: Identify low-code tags. We need to identify the tags that are related to LCSD in order to extract low-code related posts from SO discussions. To find relevant tags, we followed a similar procedure used in prior work [2, 7, 10, 161, 254, 260]. At Step 1, we identify the initial low-code related tags and call them T_{init}. At Step 2, we finalize our low-code tag list following related work [37, 277]. Our final tag list T_{final} contains 64 tags from the top 38 LCSD platforms. We discuss each step in details below.

(1) Identifying Initial low-code tags. The SO posts do not have tags like "low-code" or "lowcode". Instead, we find that low-code developers use an LCSD platform name as a tag, e.g., "appmaker" for Google Appmaker [113]. Hence, to find relevant tags, first, we compile a list of top LCSD platforms by analyzing a list of platforms that are considered as the market leaders in Gartner [258], Forrester [216], related research work [217], and other online resources like PC magazine [194]. Our compiled list contained 121 LCSD platforms, including all of our previous nine platforms from previous study [10]. Then for each of the LCSD platforms, we manually searched for the SO tags in SO. For example, we search for Oracle Apex via SO search engine and find a list of SO posts. We build a potential list of tags related with this platform based on manual inspection, such as "oracle" and "oracle-apex". Then, manually examine the metadata associated with each of these tags [1]. For example, "oracle-apex" tag's metadata says "Oracle Application Express (Oracle APEX) is a rapid Web application development tool that lets you share data and create applications. Using only a Web browser and limited programming experience, you can develop and deploy applications that are fast and secure." and "oracle" tag's metadata says "Oracle Database is a Multi-Model Database Management System created by Oracle Corporation. Do NOT use this tag for other products owned by Oracle, such as Java and MySQL.". Therefore, we select the "oracle-apex" tag for Oracle Apex platform. Not all LCSD platforms have associated SO tags; thus, they were excluded. For example, OneBlink [185] low-code platform there is no associated SO tags and thus we exclude this from our list. In order to better understand the evolution of this domain, we excluded discontinued LCSD platforms. For example, In Jan 2020, Google announced that they would no longer release new features for Google

[1]https://meta.stackexchange.com/tags

App Maker and discontinue it by 2021 [116] and so we excluded this platform from our list. Finally, we found 38 relevant SO tags from 38 platforms. The fifth and the first author participated in this step, and the complete list of the platforms and tags are available in our replication package.

So, our initial list contains 38 LCSD platforms such as: Zoho Creator [283], Salesforce [218], Quickbase [204], Outsystems [204], Mendix [171], Vinyl [259], Appian [21], and Microsoft Powerapps [203]. We thus focus on the discussions of the above 38 LCSD platforms in SO. We find one tag per LCSD platform as the name of the platform (e.g., "powerapps" for Microsoft Powerapps platform). Thus, We refer to these 38 tags as T_{init}.

(2) Finalizing low-code related tags. Intuitively, there might be more variations to tags of 38 LCSD platforms other than those in T_{init}. We use heuristics from previous related works [10, 37, 277] to find other relevant tags. First, we denote our entire SO dump data as Q_{all}. Second, we extract all the questions Q that contain any tag from T_{init}. Third, we create a candidate tag list $T_{candidate}$ using all the tags found in questions Q. Fourth, we select significantly relevant tags from $T_{candidate}$ for our LCSD discussions. Following related works [37, 254, 277], we compute significance and relevance for each tag t in $T_{candidate}$ with respect to Q (our extracted questions that has T_{init} tag) and Q_{all} (i.e., our data dump) as follows,

$$(Significance)\ \ S_{tag} \ = \ \frac{\#\ of\ ques.\ with\ the\ tag\ t\ in\ Q}{\#\ of\ ques.\ with\ the\ tag\ t\ in\ Q_{all}}$$

$$(Relevance)\ \ R_{tag} \ = \ \frac{\#\ of\ questions\ with\ tag\ t\ in\ Q}{\#\ of\ questions\ in\ Q}$$

A tag t is significantly relevant to LCSD if the S_{tag} and R_{tag} are higher than a threshold value. We experimented with a wide range of values of S_{tag} = {0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35} and R_{tag} = {0.001, 0.005, 0.010, 0.015, 0.020, 0.025, 0.03}. Figure 3.2 shows the total number of recommended vs relevant tags from our 49 experiments. It shows that as we increase S_{tag} and R_{tag} the total number of recommend tags decreases. For example, we find that for S_{tag}=.05 and R_{tag} = 0.001 the total number of recommended tags is 61 which is highest. However, not all of the suggested tags are LCSD-related. For instance, according to our significance and relevance analysis, tags such as "oracle-xe", "ems", "aura-framework", "power-automate" etc are frequently correlated with other LCSD platform tags, although they do not contain low-code-related discussions. After manually analysing these 61 tags we find that 26 tags are relevant to LCSD-related discussions. So, for the lowest S_{tag} = 0.3 and R_{tag} = 0.001 we find 26 additional LCSD-related tags. These values are consistent with related work [7, 10, 37, 254]. The final tag list T_{final} contains 64 significantly relevant tags. So, after combining with out initial taglist, i.e., T_{init}, our final tag list T_{final} contains 64 significantly relevant LCSD-related tags which are:

{ apex-code, lotus-notes, domino-designer-eclipse, visualforce, salesforce-chatter, apex, salesforce-service-cloud, simple-salesforce, salesforce-ios-sdk, apex-trigger, oracle-apex-5, salesforce-lightning, salesforce-communities, oracle-apex-5.1, servicenow-rest-api, powerapps-formula, salesforce-marketing-cloud, powerapps-selected-items, powerapps-modeldriven, powerapps-collection, powerapps-canvas, oracle-apex-18.2, lwc, salesforce-development, oracle-apex-19.1, oracle-apex-19.2, outsystems, appian, quickbase, powerapps, oracle-apex, salesforce, zoho, mendix, servicenow, pega, retool, vinyl, kissflow, bizagi, neutrinos-platform, rad, joget, filemaker, boomi, open-text, tibco, webmethods, conductor, temenos-quantum, shoutem, oracle-cloud-infrastructure, amazon-honeycode, convertigo, lotus-domino, genero, genesis, gramex, processmaker, orocrm, slingr, unqork, uniface, structr}

Step 3: Extracting low-code related posts. An SO question can have at most five tags, and we consider a question as low-code related question if at least one of its tag is in our chosen tag list T_{final}. Based on our T_{final} tag set, we found a total of 27,880 questions from our data dump. SO has a score-based system (up-vote and down-vote) to ensure the questions are in proper language with necessary information (code samples and error messages), not repeated, off-topic or incorrectly tagged. Here is an example for a question with score "-4" where a practitioner is making an API related query in Powerapps($Q_{61147923}$)[2] platform. However, it is not clear what the practitioner is asking as the question is poorly written and without any clear example. So, in order to ensure good quality discussions, we excluded questions that had a negative score. Following previous research [37, 43, 215, 254], we also excluded unaccepted answers and only considered accepted answers for our dataset. Hence, our final dataset B contained 37,766 posts containing 67.4% Questions (i.e., 26,763) and 32.6% Accepted Answers (i.e., 11,010).

To ensure that our final taglist T_{final} comprises discussions relating to low-code software development, we randomly select 96 questions from our dataset that are statistically significant with a 95% confidence level and 10 confidence interval. First and third authors contributed to this manual analysis, and after manual analysis, we found that 98% of questions from our selected taglist contain low-code platform-related discussion, with only two questions containing discussion that is not particularly related to low-code platforms. For instance, question $Q_{59402662}$ includes the tag "appian", yet the question body describes only about a MySQL database performance-related issue on the Azure platform. Similarly, the question $Q_{19289762}$ contains the "apex-code" tag, but exclusively discusses AWS cloud authentication signature-related issues in its problem description.

3.3.2 Topic Modeling

We produce LCSD topics from our extracted posts in three steps: (1) Preprocess the posts, (2) Find optimal number of topics, and (3) Generate topics. We discuss the steps below.

Step 1. Preprocess the posts. For each post text, we remove noise following related works [2,37,43]. First, we remove the code snippets from the body, which is inside <code></code> tag, HTML tags such as (<p></p>, <a></a>, <li></li> etc), and URLs. Then we remove the stop words such as "the", "is", "are", punctuation marks, numbers,

[2] Q_i and A_i denote a question Q or answer A in SO with an ID i

non-alphabetical characters using the stop word list from MALLET [168], NLTK [162], and our custom low-code specific (i.e., LCSD platform names) stop word list. We remove the platform's name from the dataset since, based on our experiments with LDA topic modeling for this study and our past work [10], the resultant topics sometimes tend to cluster around LCSD platforms rather than the technical challenges discussed. Thus, we remove the LCSD platform names from our dataset. After this, we use porter stemmer [205] to get the stemmed representations of the words e.g., "wait", "waits", "waiting", and "waited" - all of which are stemmed to base form "wait".

Step 2. Finding the optimal number of topics. After the prepossessing, we use Latent Dirichlet Allocation [52] and the MALLET tool [168] to find out the LCSD-related topics in SO discussions. We follow similar studies in Software engineering research using topic modeling [2,22,24,37,277]. Our goal is to find the optimal number of topics K for our dataset B so that the *coherence* score, i.e., encapsulation of underlying topics, is high. We use Gensim package [208] to determine the coherence score following previous works [213, 244]. We experiment with different values of K that range from $\{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70\}$ and for each value, we run MALLET LDA on our dataset for 1000 iterations [37]. Then we observe how the coherence score is changing with respect to K. We pick the topic model with the highest coherence score. Choosing the right value of K is important because, for smaller values of K, multiple real-world concepts merge, and for a large value of K, a topic breaks down. For example, in our result, the highest coherence score 0.50 for $K = 45$ and $K = 40$. The first, third, fourth, and fifth authors participate in the analysis and we choose $K = 45$ as it captures our underlying topics better. MALLET also uses two hyper-parameters, α and β, to distribute words and posts across the topics. Following the previous works [7, 10, 37, 39, 215, 254], we use the standard values $50/K$ and 0.01 for hyper-parameters α and β in our experiment.

Step 3. Generating topics. Topic modeling is a method of extracting a set of topics by analysing a collection of documents without any predefined taxonomy. Each document has a probability distribution of topics, and every topic has a probability distribution of a set of related words [43]. We produced 45 topics using the above LDA configuration on our dataset B. Each topic model offers a list of top N words and a list of M posts associated with the topic. In our settings, a topic consists of 30 most frequently co-related words, which represent a concept. Each post had a correlation score between 0 to 1, and following the previous work [10, 254, 260], we assign a document with a topic that it correlates most.

3.4 Empirical Study

We report the results of an empirical study by answering the following five research questions (RQ) based on our analysis of LCSD topics in our dataset.

RQ1. What topics do LCSD practitioners discuss? (Section 3.4.1)

RQ2. How do the LCSD topics evolve over time in SO? (Section 3.4.2)

RQ3. What types of questions are asked across the observed topic categories? (Section 3.4.3)

RQ4. How are the observed topic categories discussed across SDLC phases? (Section 3.4.4)

RQ5. What LCSD topics are the most difficult to answer? (Section 3.4.5)

The first two research questions (RQ1, RQ2) provide insights about what topics practitioners discuss in SO and how these topics have evolved over time. The third and fourth research questions (RQ3, RQ4) explore the types of questions in these topics and they affected different SDLC phases. At the end, we discuss the popularity and difficulty of the LCSD topics in the last research question (RQ5).

3.4.1 What topics are discussed about LCSD in Stack Overflow? (RQ1)

3.4.1.1 Motivation

The increased popularity of LCSD as a flexible and straightforward approach helps develop practical business applications. The challenges of LCSD are yet to be studied as this is a new approach to software development. SO is an established source of knowledge repository to systematically study the real-world challenges that the practitioners face. An understanding of the LCSD topics in SO developer discussions will help LCSD platform providers and researchers to have a better understanding of the underlying prevalent issues, which can then help guide efforts to improve the quality of LCSD platforms.

3.4.1.2 Approach

We applied LDA topic modeling to our LCSD-related discussion in SO. We get 45 low-code related topics from our LDA topic modeling, as discussed in Section 4.3. We use card sorting [102] to label these topics following previous works [2, 7, 37, 215, 277]. In open card sorting, there is no predefined list of labels. Following related works [2, 10, 37, 254], we label each topic by analyzing the top 30 words for the topic and a random sample of at least 20 questions that are assigned to the topic. Four of the authors participated in the labeling process in group sessions (first, third to fifth). Each author assigns a label to each topic and discusses it until there is an agreement. The authors reached an agreement after around 10 iterations of meetings over Skype and email and labeled the 45 topics from the LDA output.

After this initial labeling, we merged a few topics because they contained similar discussions with different vocabularies. For example, we merged topic 36 and 43 into Dynamic form controller because both topics contained discussions related to forms with a predefined list of values, dynamically changing the fields (or options) of forms

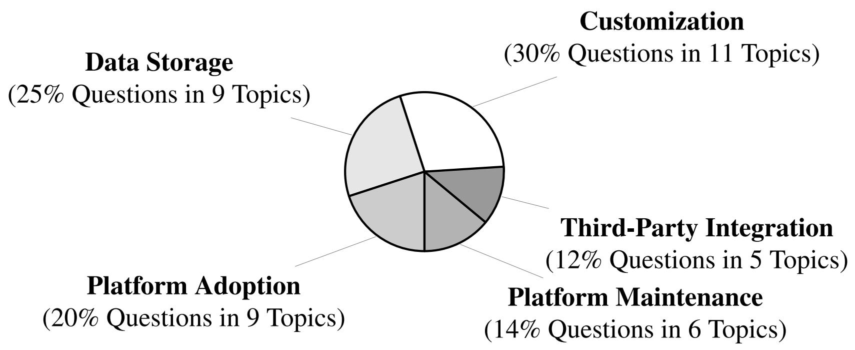


Figure (3.3) Distribution of Questions and Topics per Topic Category

values based on users' actions or previous selections. Similarly, we merged topic 2 and 19 to DB Setup & Migration. In the end, we obtained 40 distinct LCSD-related topics.

After the labeling of the topics, we revisited the labels in an attempt to find any clusters/groups among the topics. For example, Date & Time Manipulation, Formatted Data Parsing, and Pattern Matching topics are related, and thus, they are grouped under the General Programming category. We repeated this process multiple times to find increasingly higher categories. For example, we found another category called Dynamic Content which contained two topics Dynamic Data Binding and Dynamic Data Filtering. We then put these two categories under called Business Logic Implementation. This higher abstraction helped us to place other topics related to implementing business logic under this category. Following a similar strategy, we put this Business logic implementation under the Customization category, which discussed customizing applications. For example, under Customization, there were a category called UI which contained Dynamic Layout, and Script category, which contained topics such as Dynamic Page Layout, Dialog Box Manipulation, Window Style Manipulation, and Dynamic Form Controller. The entire process of creating this hierarchy of topic categories took multiple iterations and revisions. We created a coding guideline for creating the taxonomy of topics to ensure consistency and reproducibility. We share the coding guide with our replication package.

3.4.1.3 Results

After labeling and merging, we find 40 LCSD-related topics. Then after grouping these topics into higher categories, we find five high-level categories: (1) Customization, (2) Data Storage, (3) Platform Adoption, (4) Platform Maintenance, and (5) Third-Party Integration . Figure 4.3 shows the distribution of topics and questions into these five categories. Among these categories, Customization has the highest coverage of questions and topics (30% Questions in 11 Topics), followed by Data Storage (25% Questions in 9 Topics), Platform Adoption (20% Questions in 9 Topics), Platform Maintenance (14% Questions in 6 Topics), Third-Party Integration (12% Questions in 5 Topics).

Figure 4.4 shows the 40 LCSD topics sorted by numbers of posts. A post means an LCSD-related question or an accepted answer in our case. As discussed in Section 4.3.1, our dataset has total 37,773 posts containing 26,763 questions and 11,010 accepted answers. The topic with the highest number of posts is placed on top of the list. On

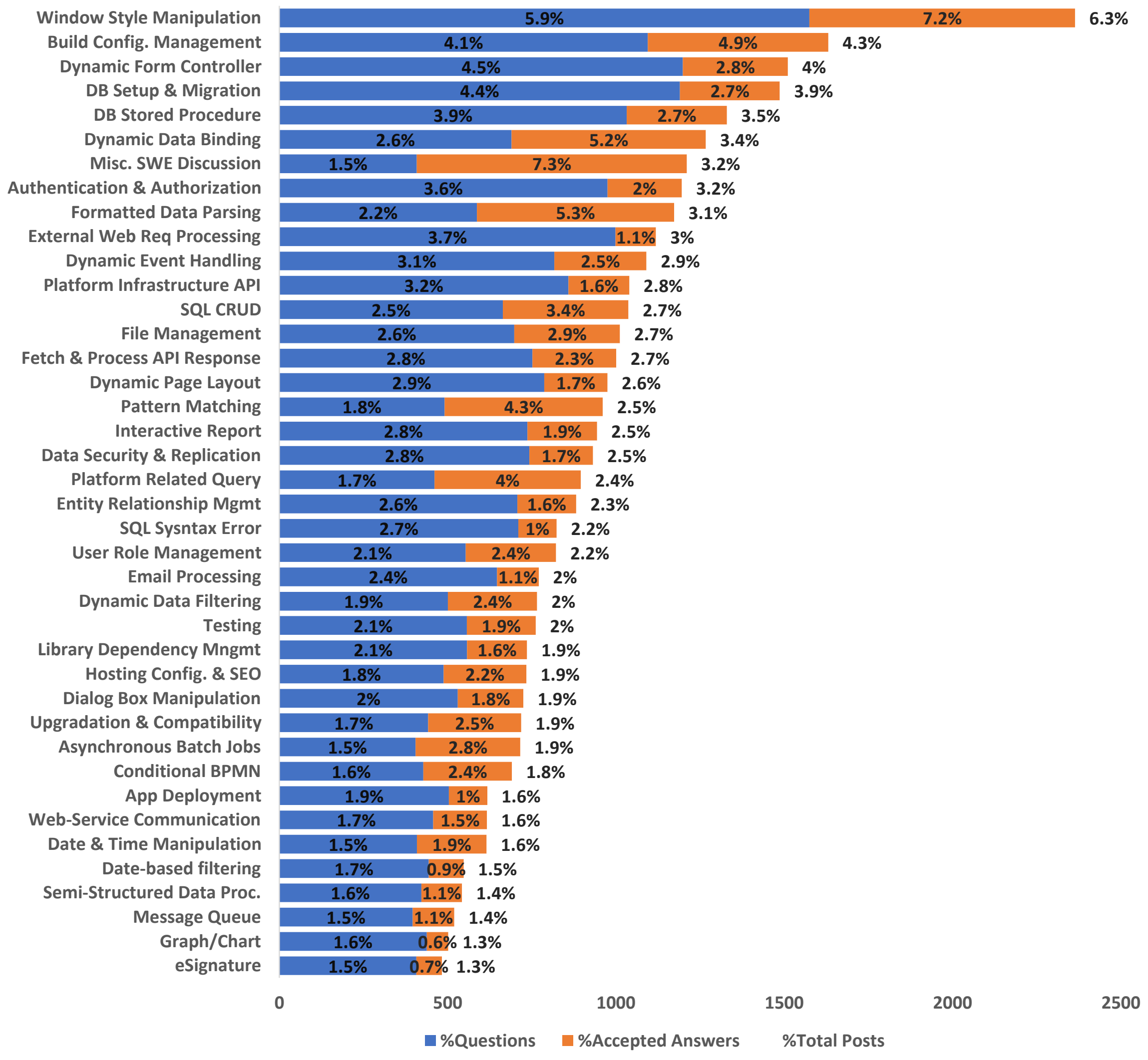


Figure (3.4) Distribution of questions by low-code topics by total number of posts (Number of Questions + Number of Accepted Answers)

average, each topic has 944 posts (question + accepted answer). The topic Window Style Manipulation has the highest number of posts (6.3%) with Questions 5.9% of total questions and 7.2% total accepted answers. On average, each topic has around 669 questions.

Figure 4.5 provides a taxonomy of 40 LCSD related topics into five categories. The topics are organized in descending order of the number of questions. For example, the Customization category has the highest number of questions, followed by Data Storage. Each category may have some sub-categories of topics. For example, the Customization category has two sub-categories: UI and Business Logic. The topics under each sub-category can further be grouped into multiple sub-sub-categories. For example, the UI sub-category has 4 topics grouped into Script and Dynamic Layout sub-sub-categories. Each sub-category, sub-sub-categories, and topics are also presented

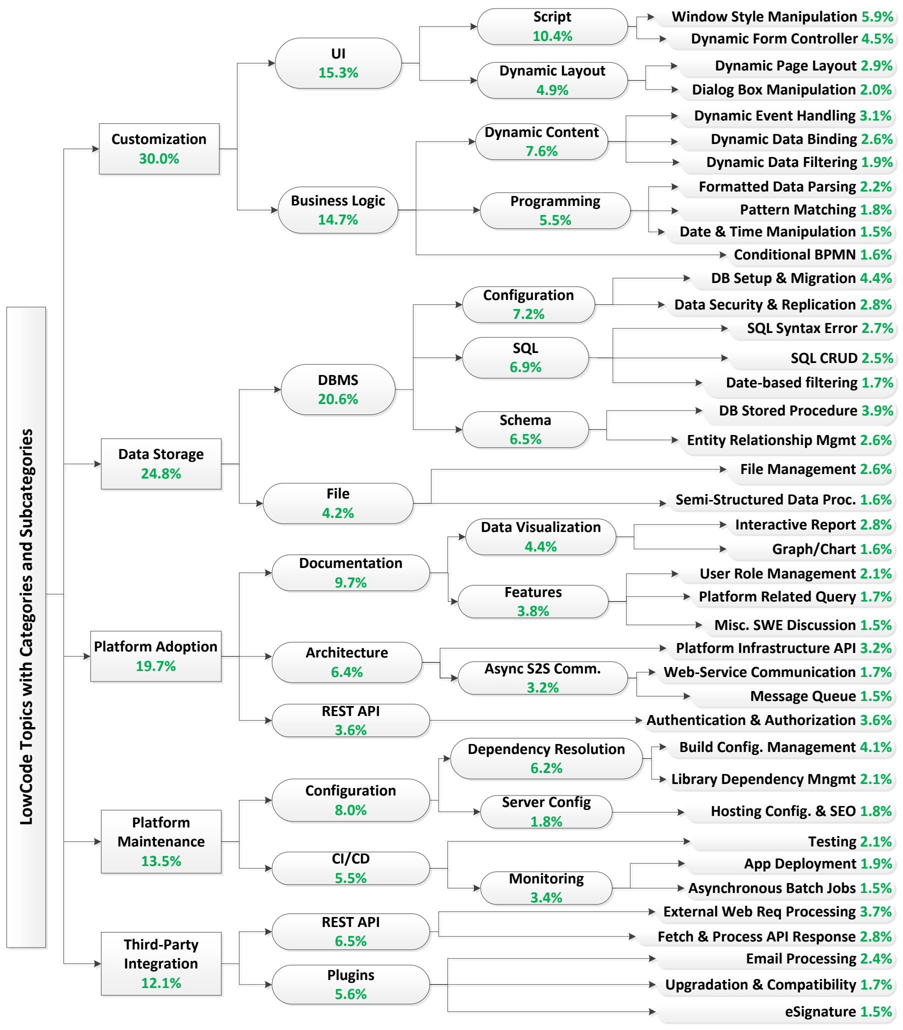

LowCode Topics with Categories and Subcategories
Customization 30.0%
UI 15.3%
Script 10.4%
Window Style Manipulation 5.9%
Dynamic Form Controller 4.5%
Dynamic Layout 4.9%
Dynamic Page Layout 2.9%
Dialog Box Manipulation 2.0%
Business Logic 14.7%
Dynamic Content 7.6%
Dynamic Event Handling 3.1%
Dynamic Data Binding 2.6%
Dynamic Data Filtering 1.9%
Programming 5.5%
Formatted Data Parsing 2.2%
Pattern Matching 1.8%
Date & Time Manipulation 1.5%
Conditional BPMN 1.6%
Data Storage 24.8%
DBMS 20.6%
Configuration 7.2%
DB Setup & Migration 4.4%
Data Security & Replication 2.8%
SQL 6.9%
SQL Syntax Error 2.7%
SQL CRUD 2.5%
Date-based filtering 1.7%
Schema 6.5%
DB Stored Procedure 3.9%
Entity Relationship Mgmt 2.6%
File 4.2%
File Management 2.6%
Semi-Structured Data Proc. 1.6%
Platform Adoption 19.7%
Documentation 9.7%
Data Visualization 4.4%
Interactive Report 2.8%
Graph/Chart 1.6%
Features 3.8%
User Role Management 2.1%
Platform Related Query 1.7%
Misc. SWE Discussion 1.5%
Architecture 6.4%
Platform Infrastructure API 3.2%
Async S2S Comm. 3.2%
Web-Service Communication 1.7%
Message Queue 1.5%
REST API 3.6%
Authentication & Authorization 3.6%
Platform Maintenance 13.5%
Configuration 8.0%
Dependency Resolution 6.2%
Build Config. Management 4.1%
Library Dependency Mngmt 2.1%
Server Config 1.8%
Hosting Config. & SEO 1.8%
CI/CD 5.5%
Testing 2.1%
Monitoring 3.4%
App Deployment 1.9%
Asynchronous Batch Jobs 1.5%
Third-Party Integration 12.1%
REST API 6.5%
External Web Req Processing 3.7%
Fetch & Process API Response 2.8%
Plugins 5.6%
Email Processing 2.4%
Upgradation & Compatibility 1.7%
eSignature 1.5%

Figure (3.5) A taxonomy of LCSD topics with sub-categories.

in descending order of the number of questions.

We discuss the five categories and the 40 topics in detail below.

Customization Topic Category. Customization is the largest topic category in terms of the number of topics and percentage of questions. Out of the 40 topics, 11 topics belong to the Customization category, with around 30% of questions in our dataset. These topics contain discussions about implementing business logic, customizing UI, input and form data validation, general programming-related query to implement some features, etc. This category has two sub-categories: (1) UI contains discussion about customizing the UI, dynamically changing window components, interactive dialog boxes, and (2) Business Logic contains discussion about different programming customization-related queries, dynamically binding UI elements to backend data.

• **UI Sub-Category** contains 15% questions and four topics divided in two sub-sub-categories: (1) Script contains discussion about manipulation of text widgets, formatting components, and (2) Dynamic Layout is about hiding and moving components, showing popups.

The **Script** sub-sub-category contains 10.4% questions and has two topics: (1) Topic *Window Style Manipulation (5.9%)* concerns about manipulating the style of the HTML documents such as adding/removing margins/padding (e.g., $Q_{36503030}$), adding links, manipulating navigation bar and embedded views (e.g., $Q_{30453620}$). (2) Topic *Dynamic Form Controller (4.5%)* are about dynamic form, i.e., forms with predefined list of values (e.g., $Q_{64373454}$), multi select content (e.g., $Q_{39318510}$), changing forms option based on previous selection (e.g., $Q_{43725028}$). The **Dynamic Layout** sub-sub-category covers 4.9% questions and has two topics: (1) Topic *Dynamic Page Layout (2.9%)* contains discussion about UI (i.e. page) customization (e.g., $Q_{65964413}$), pagination in $Q_{4536018}$, hiding or moving element based on some user action or an event (e.g., $Q_{13231072}$)., (2) Topic *Dialog Box Manipulation (2.0%)* is about manipulating dialog box (e.g., pop up/ modals) such as hiding them in $Q_{49804455}$, close them in $Q_{55513527}$, show popup, refresh web-page (e.g., $Q_{60606986}$, $Q_{21701437}$).

• **Business Logic Sub-Category** contains 14.7% questions and 7 topics in two sub-sub-categories: (1) Programming is about discussion related to different programming-related questions and data access, and (2) Dynamic Content is about discussions related to dynamically querying data from different data sources, dynamically changing the web-page content. The Business Logic sub-category contains one topic *Conditional BPMN* that does not belong to any sub-sub-category. Topic Conditional BPMN (1.6%) contains LCSD platform's application customization related discussions on Business Process Model and Notation (BPMN) (e.g. $Q_{38265464}$) and conditional logic features (e.g., $Q_{66335289}$, $Q_{65838553}$).

The **Dynamic Content** sub-sub-category contains 7.6% of questions and has three topics: (1) Topic *Dynamic Event Handling (3.1%)* discusses about different JavaScript related issues such as JavaScript feature not working (e.g., "JavaScript promise is not working" in $Q_{65550370}$), browser compatibility, JS event initialization issue (e.g., $Q_{64507615}$) etc. (2) Topic *Dynamic Data Binding (2.6%)* is about discussions related to the design of forms with predefined values (e.g., $Q_{45051098}$), the implementation of multi-select, customized drop-down values, form validation (e.g., $Q_{51115573}$), changing content of one field based on the value of other field in $Q_{47652192}$. (3) Topic *Dynamic Data Filtering (1.9%)* contains business logic customization related discussion based on advanced filtering criteria and querying multiple tables. (e.g., "Find Records Based on the Contents of Records in a Related Table?" in $Q_{20665253}$ and "find the latest record grouped by name in layout" in $Q_{38128584}$).

The **Programming** sub-sub-category contains 5.5% of questions and has three topics: (1) Topic *Formatted Data Parsing (2.2%)* is about programming related discussion on parsing formatted data, i.e., JSON (e.g., $Q_{50184058}$, $Q_{44803257}$), XML (e.g., $Q_{13785513}$), array of objects (e.g., $Q_{66744874}$) etc. (2) Topic *Pattern Matching (1.8%)* topic concerns programming related discussions about searching and modifying strings by pattern matching using regular expression (e.g., "How do I search for an Exact phrase" in $Q_{51258323}$, "Regex pattern to replace html in a given text string" in $Q_{48251198}$). (3) Topic *Date & Time Manipulation (1.5%)* contains programming discussions about date-time manipulation like conversion of formatted string from data-time in $Q_{51714301}$, calculation of difference between date-time (e.g., $Q_{59230493}$) and timezone, time conversion (e.g., "How to convert a Date value to unix timestamp?" in $Q_{60601201}$).

Data Storage Topic Category. Data Storage Category is the second largest topic category with a total of 9 topics and around 25% of the questions of our dataset. This topic category contains discussions on database management and file storage. It contains two sub-categories: (1) DBMS is about discussion related to database setup, migration, DB query, (2) File concerns storing and retrieving files (i.e., images, CSV files, etc.).

• **DBMS Sub-Category** contains around around 20.6% questions with seven topics under three sub-sub-categories: (1) Configuration contains discussions about database setup, database connection, DB data security, (2) SQL contains discussion about SQL query, (3) Schema is about database schema design (i.e., Primary key, foreign key design), different stored procedure.

Configuration sub-sub-category contains 7.2% questions and two topics: (1) Topic *DB Setup & Migration (4.4%)* topic is about connecting applications to different vendor databases (i.e., MySQL, Postgres, Oracle etc.) (e.g., "is ODBC Firebird connection possible?" in $Q_{28251836}$), DB users (e.g., $Q_{58815776}$), issues about different database versions, data migration to LCSD platform (e.g., "How to add External data source into MySQL?" in $Q_{28251836}$ or $Q_{22626970}$). (2) Topic *Data Security & Replication (2.8%)* topic concerns about discussion related to data security (e.g., encryption and decryption in $Q_{1567252}$), accessing stored of database file (e.g., $Q_{5730482}$), data backup or replication (e.g., $Q_{10997987}$) etc. **SQL** sub-sub-category contains 6.9% questions and three topics: (1) Topic *SQL Syntax Error (2.7%)* discusses about errors in syntax in different SQL query and stored procedure. For example, there are questions such as "I'm getting error while creating a procedure in pl/sql" in $Q_{63990287}$, "SQL parsing fails, not sure what the issue is?" in $Q_{8298486}$. (2) Topic *SQL CRUD (2.5%)* is about database Create, read, update and delete (i.e., CRUD) related queries (e.g., $Q_{22712624}$), and advanced queries too, such as inner join, nested join, aggregate (e.g., "SQL Query: JOIN Three tables then Not showing the results after joining the 3rd table" in $Q_{22712624}$). This topic also contains discussion about Object query language, which is a high-level wrapper over SQL in $Q_{64812548}$. (3) Topic *Date-based filtering (1.7%)* contains database query related discussion specially for date-time based filtering (e.g., $Q_{52389335}$), i.e., monthly/quarterly query, time-based data grouping etc. For example, there are questions such as How to count total amount of value by day (e.g., $Q_{65142062}$). **Schema** sub-sub-category contains 6.5% questions and two topics: (1) Topic *DB Stored Procedure (3.9%)* is about database schema and advanced database related discussion on stored procedure (e.g., support of triggers in LCSD platform, $Q_{11799577}$, $Q_{37810803}$). (2) Topic *Entity Relationship Mgmt (2.6%)* concerns about discussion on advanced database schema design (e.g., "How to automatically insert foreign key into table after submit in [LCSD platform]", $Q_{66968187}$) and database discussion to automatically update database (e.g., "Auto increment item in Oracle APEX" in $Q_{61961348}$).

• **File Sub-Category** contains 4.2% questions with two topics: (1) Topic *File Management (2.6%)* contains discussion of file management, i.e., storing and processing files, renaming it in $Q_{56414466}$, converting files from one format to another in $Q_{45796962}$, handling image files (e.g, $Q_{34203211}$). (2) Topic *Semi-Structured Data Proc. (1.6%)* is about different programming related discussion on processing, modifying and storing semi-structured data files, i.e., XML, CSV files. For example, there are questions such as Fetch CSV file columns dynamically Using [platform] package in $Q_{66310875}$.

Platform Adoption Topic Category. A total of nine topics belong to the Platform Adoption category with around 20% questions. The nine topics belong to three sub-categories: (1) Documentation contains LCSD platform's feature-related discussions and how to use those features, (2) Architecture concerns about what type of software development architecture (e.g., client-server communication) is supported by the LCSD platforms, (3) REST API contains LCSD platform's RESTful APIs.

• **Documentation Sub-Category** contains 9.7% questions and five topics. Four of the topics fall under two sub-sub categories: (1) Data Visualization contains discussion related to interactive reports and graphs, (2) Features is about LCSD platform provides features such as user's role management, support on SDLC management. Topic *Misc. SWE Discussion (1.5%)* concerns about discussions related to general software engineering such as Unix Threading (e.g., $Q_{30530873}$), Object-oriented programming (e.g., Q_{314241}), auto scaling, ambiguous documentation in $Q_{10348746}$.

Data Visualization sub-sub-category contains 4.4% questions and has two topics: (1) Topic *Interactive Report (2.8%)* is about data visualization and interactive data reports. It contains developers' discussions about how they can use different platform features for customized reports. For example, "using jquery hide column heading when no data in column in interactive report in [platform]" in $Q_{53294659}$. (2) Topic *Graph/Chart (1.6%)* discusses about platform's support and documentation request to draw different graphs (e.g., $Q_{41257691}$) and charts using stored data. For example, "How to overlay a line plot over a bar graph in [platform]?" in $Q_{28727869}$ **Features** sub-sub-category contains 3.8% questions and has two topics: (1) Topic *User Role Management (2.1%)* contains discussion about different user role management features (i.e., administrators and regulators) provided by the LCSD platforms. It discusses about user's profile management (e.g., $Q_{66853056}$), user group and access-level (e.g., $Q_{35457840}$). (2) Topic *Platform Related Query (1.7%)* contains general discussion about LCSD platforms such as comparison of features between different platforms (e.g., "How is [platform A] better than [platform B] in BPM?" in $Q_{39127918}$), Agile and RAD development support (e.g., $Q_{2512396}$), performance of a specific feature of a platform in $Q_{6068882}$.

• **Architecture Sub-Category** contains 6.4% questions and two topics and one sub-sub category: (1) Async S2S Comm contains discussion related to distributed applications with service to service communication. Topic *Platform Infrastructure API (3.2%)* contains cloud-based REST API from the LCSD platforms to configure and utilize different platform features, connect to other services or data sources via connectors or cloud REST APIs. For example, the questions are about how [platform] apps portal integration with [DB] On-premise in $Q_{63688934}$, "Change Shape OCI instance with Ansible" in $Q_{60511836}$.

Async S2S Comm sub-sub-category contains 3.2% questions and has two topics: (1) Topic *Web-Service Communication (1.7%)* contains discussions about micro-service architecture, service to service communication via web service description language (e.g., $Q_{16278661}$, $Q_{2567466}$), HTTP REST message in $Q_{58689313}$, Windows Communication Foundation (e.g., $Q_{36849686}$). (2) Topic *Message Queue (1.5%)* is about discussion about different asynchronous service-to-service data exchange mechanisms such as using a message queue. It generally contains discussions about micro-service design patterns and producer and consumer mechanisms (e.g., $Q_{41640881}$) for data exchange. For example, "How to know who is connected to a [Platform] EMS Queue" in $Q_{66999418}$, $Q_{56334001}$.

REST API sub-category contains 3.6% questions and has one topics: (1) Topic *Authentication & Authorization (3.6%)* contains discussion about LCSD platforms support on different standard authentication and authorization protocol such as OAuth2 (e.g., $Q_{30475542}$), SAML (e.g., $Q_{23624206}$), access token (e.g., "access token in android [Platform] sdk" in $Q_{32943204}$).

Platform Maintenance Topic Category. We find 6 topics and 13.5% questions in Platform Maintenance category. It has two sub-categories: (1) Configuration contains discussion on LCSD platforms library and build configuration, (2) CI/CD. is about discussion related to DevOps tasks such as continuous integration and continuous delivery, testing etc.

• **Configuration Sub-Category** contains 8.0% questions and three topics under two sub-sub categories: (1) Dependency Resolution is about LCSD platforms server's library dependency management, (2) Server Configuration is about LCSD platform's servers configuration and hosting settings such as SSL configuration.

Dependency Resolution sub-sub-category contains 6.2% questions and has two topics: (1) Topic *Build Config. Management (4.1%)* contains discussion about system build configuration and external library management-related issues in $Q_{48727452}$. This topic also contains discussion about compilation failure (e.g., $Q_{29243987}$), library dependency, build path not configured properly (e.g., $Q_{57015131}$) etc. (2) Topic *Library Dependency Mngmt (2.1%)* is about the library and dependencies of the system (e.g. $Q_{23471590}$, $Q_{62872836}$), server configuration, different library version compatibility issues in $Q_{60050869}$. **Server Config.** sub-sub-category contains 1.8% questions and has one topic: (1) Topic *Hosting Config. & SEO (1.8%)* is about discussions about LCSD platforms support on server configuration, i.e., configuring SSL certificate (e.g., "[platform] client ignoring expired certificate" in $Q_{55044903}$), LCSD platform's support on making the application accessible and index-able (e.g., $Q_{34860991}$).

• **CI/CD Sub-Category** contains 5.5% questions and three topics. Two of the topics fall under one sub-sub category:(1) Monitoring is about the discussion on monitoring the deployed applications and scheduled job status. Topic

Testing (2.1%) contains discussions about LCSD platforms support on testing and test coverage. For example, "How to know overall code coverage of multiple test classes?" in $Q_{67724447}$, How to write a test class as in $Q_{50586482}$.

Monitoring sub-sub-category contains 3.4% questions and has two topics: (1) Topic *App Deployment (1.9%)* discusses about the LCSD platform's CI/CD features such as incrementally updating the application code in $Q_{39045129}$, deployment packages as in $Q_{4813597}$, monitoring the changes in the application code (e.g., $Q_{61938011}$). (2) Topic *Asynchronous Batch Jobs (1.5%)* contains discussions about LCSD platforms' support for monitoring and scheduling asynchronous batch jobs and scheduled tasks. For example, "How to get your failing batch records?" in $Q_{11068830}$, "How can I schedule apex to run every 30 seconds?" in $Q_{17143633}$.

Third-Party Integration Topic Category is smallest topic category based on number of questions (12.1% questions). It has five Topics. Four of its topics fall under two sub-categories: (1) REST API contains discussion related to RestFul API communication with third-party services, (2) Plugins is about discussion about external plugins and APIs that are supported by the LCSD platforms.

• **REST API Sub-Category** contains 6.5% questions and two topics: (1) Topic *External Web Req Processing (3.7%)* contains discussion about integrating 3rd party REST APIs, processing and parsing external requests such as "Connect to [Platform] REST API with [Service] data integration" in $Q_{51865601}$, $Q_{46033973}$. (2) Topic *Fetch & Process API Response (2.8%)* contains discussions about making HTTP request to remote servers (e.g., "REST http post method - what does -d mean in a curl?" in $Q_{65877037}$), analyzing and processing the response, handling web security issues (e.g., CORS policies in $Q_{60270574}$).

• **Plugins Sub-Category** contains 5.6% questions and three topics: (1) Topic *Email Processing (2.4%)* discusses about processing automating emailing in $Q_{65626477}$, sending formatted HTML email (e.g, "Send HTML email using [platform]" in $Q_{41546887}$), forwarding emails in $Q_{18234790}$ etc. (2) Topic *Upgradation & Compatibility (1.7%)* contains discussion about application version migration as in $Q_{16894245}$, upgradation and compatibility issues of different plugins used in low-code applications (e.g., $Q_{18231293}$, $Q_{49017642}$). (3) Topic *eSignature (1.5%)* contains discussion about different issues and customization for electronic signature of documents, i.e., docusign about collecting user's agreement/permission for sales or account opening. For example, there are questions such as "Auto Add Document to DocuSign [Platform] Using Custom Button" in $Q_{34804072}$, $Q_{27512874}$.

RQ1. What topics are discussed about LCSD in SO? We found 40 Topics organized into five high-level categories. The Customization category (30%) has the highest number of questions, followed by Data Storage (25%), Platform Adoption (20%), Platform Maintenance (14%), and Third-Party Integration (12%). Window Style Manipulation from the Customization category has the highest number of questions (5.9%) followed by build Configuration (4.1%) Management from the Platform Maintenance category. Our studies reveal that low-code practitioners struggle with RESTful API Integration, configuration and maintenance of the platforms. We also observed that proper documentation could have mitigated these challenges to a great extent.

3.4.2 How do the LCSD topics evolve over time? (RQ2)

3.4.2.1 Motivation

Our analysis of RQ1 finds that LCSD topics are diverse. For example, the Customization topic category contains discussions about developing and customizing the application, and Platform Adoption and Platform Maintenance topic contains discussions related to different features provided by the LCSD platform providers. The platforms for LCSD continue to evolve, as do the underlying topics and question types. We study the evolution of these topics and question types to understand better the evolution and adoption of LCSD development and its community. This analysis will provide valuable insights into the LCSD community and help identify if any topic needs special attention.

3.4.2.2 Approach

Following related studies [254], we study the absolute and relative impacts of each of our observed five LCSD topic categories as follows.

Topic Absolute Impact. We apply LDA topic for our corpus C and get K topics (t_1, t_2,........,t_k). The absolute impact metric for a topic t_k in a month (m) is defined as:

$$Impact_{absolute}(t_k; m) = \sum_{p_i=1}^{D(m)} \theta(p_i; t_k) \tag{3.1}$$

Here the $D(m)$ is the total number of SO posts in the month m and $\theta(p_i; t_k)$ denotes the possibility for a post (p_i) belonging to a topic t_k.

From our topic modeling, we found 40 topics that were categorized into five high topic categories, i.e., *Customization, Data Storage, Platform Adoption, Platform Maintenance, Third-Party Integration*. Now, we further refine the equation for absolute impact for LCSD topics to find absolute impact metrics for a topic category (TC_j) for a month

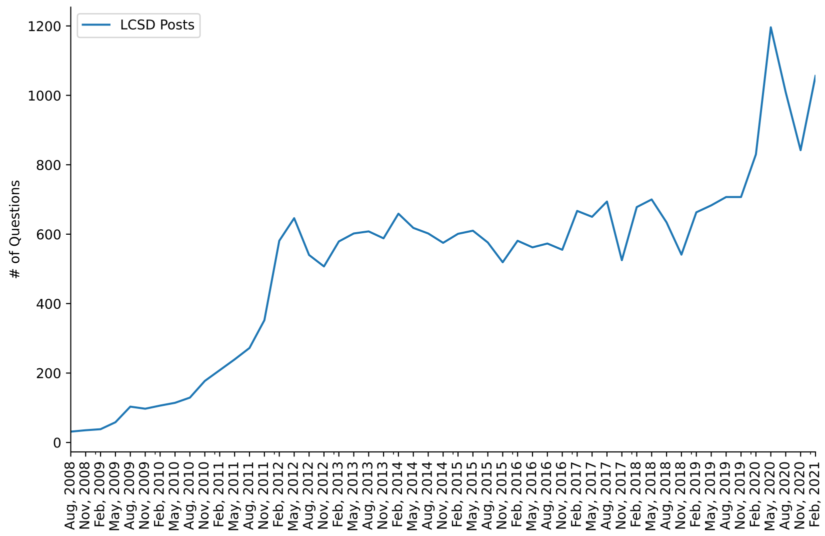


Figure (3.6) The evolution of overall LCSD-related discussions over time.

m as follows:

$$Impact_{absolute}(TC_j; m) = \sum_{t_k}^{TC_j} Impact_{absolute}(t_k; m), 0 < j < TC \tag{3.2}$$

Topic Relative Impact. Related impact metric signifies the proportion of posts for a particular LCSD topic t_k relative to all the posts in our corpus C for a particular month m. Following related studies [254], we compute the related impact metrics for LCSD topics. We use the following equation to compute the metric for a topic t_k in a month m as follows:

$$Impact_{relative}(t_k, m) = \frac{1}{|D(m)|} \sum_{p_i=1}^{\theta} (p_i; t_k), 1 \le i \le C \tag{3.3}$$

Here $D(m)$ denotes the total number of posts that belongs to a topic t_k for a particular month m. Here δ denotes the probability of a particular post p_i for our Corpus C belonging to a particular topic t_k.

Similar to the absolute impact, we refine the equation to compute the relative impact on LCSD topic categories as follows:

$$Impact_{relative}(TC; m) = \sum_{t_k}^{TC} Impact_{relative}(t_k; m) \tag{3.4}$$

Here TC donates one of our five topic categories and topics that belong to each topic category.

3.4.2.3 Result

Figure 3.6 depicts the progression of overall LCSD-related conversation using the absolution impact equation from our extracted dataset between 2008 to 2021. Additionally, it demonstrates that the peaks of LCSD-related talks occurred in mid-2020 (i.e., around 400 questions per month). It also reveals that LCSD-related discussion has been gaining

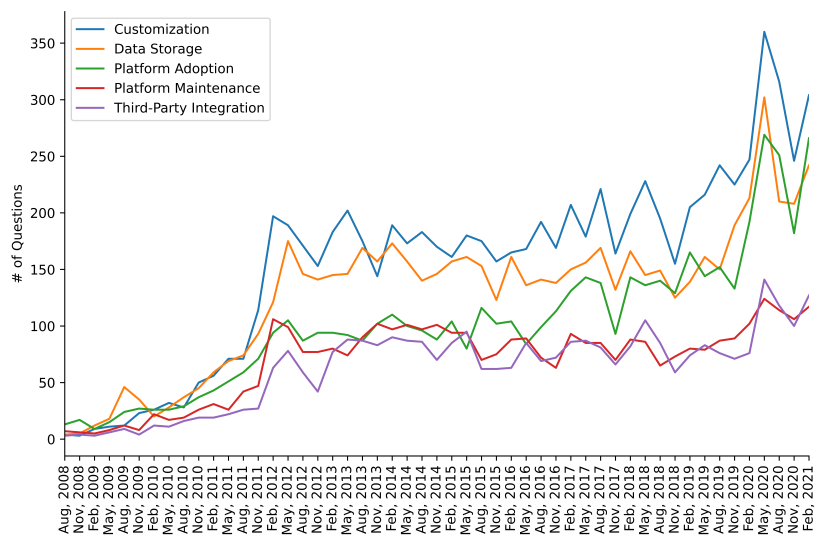


Figure (3.7) The absolute impact for LCSD topic categories over time.

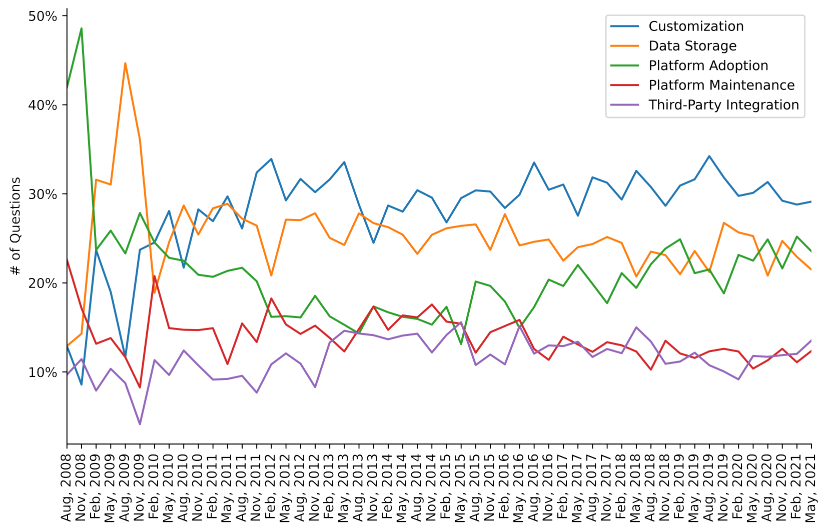


Figure (3.8) The relative impact for LCSD topic categories over time.

popularity since 2012. In the section below, we provide a more extensive explanation for the minor spikes in the Figure 3.6.

We observe that in the early days (i.e., 2009), Platform Adoption along with Data Storage topic category has more questions compared to Customization. Customization topic category starts to get a dominant position from mid (i.e., August) of 2011 over Platform Adoption, and it remains the dominant topic till the end of 2021. The number of questions in the Customization topic category gradually increased over time, from August 2011 (23) to March 2012 (81) to May 2020 (99). Data Storage topic category briefly exceeds Customization Category during August 2013, but it mostly holds a dominant position other times compared to Platform Adoption topic category. On the other hand, Platform Maintenance and Third-Party Integration exhibits very similar evolution over the whole time.

We further look into the Figure 3.7 and see there are mainly two significant upticks in the number of posts about LCSD. The first one is between August 2011 to May 2012, when there is a sharp increase in the number of questions for almost all topic categories, especially for Customization and Data Storage Category. By this Salesforce [218] LCSD platform-related discussion becomes quite popular in SO, and around that time, it ranks very high as a CRM platform. The second increase in posts is between February 2020 and August 2020. During this time of the Covid19 pandemic, many businesses began to operate online and there is a significant uptick in the number of questions in the Customization category, followed by Data Storage and Platform Adoption Moreover, there is an uptick in the number of questions on building a simple app to collect and report data, especially the Salesforce platform. There is an increase in the Platform Adoption topic category between mid-2016 to mid-2017. During this time Oracle Apex platform released version 5.0, and there is an uptick of questions regarding different features such as interactive grid in $Q_{43316233}$, drag and drop layout in $Q_{45818292}$. Now we provide a detailed analysis of each of the five topic categories.

Customization This is the biggest topic category with 11 topics. From 2008 to mid of 2011, all of these topics evolve homogeneously. From the mid of 2011 to the first Quartile of 2012, Dynamic Page Layout topic becomes dominant. “How to get fields in page layout” in $Q_{7256190}$, issues with page layout in different LCSD platforms (e.g., $Q_{7421985}$). From the end of 2012 to 2017, Window Style Manipulation topic remains most dominant. “Passing values from child window to parent window which is on another domain?” in $Q_{16463602}$, view related issues $Q_{15715645}$. From the end of 2017 to the end, our dataset Dynamic Form Controller topic remains the most dominant.

Data Storage Category From mid-2015, Database Setup & Migration topic becomes the most dominant topic in this category and has some high spikes during the pandemic and mid of 2017. For instance, there are queries like “Using Jenkins for OCI database migration” in $Q_{62217796}$ and “Almost all the cloud service providers have 99.95% of data available on the cloud. What will happen if the whole region sinks in an earthquake?” in $Q_{62102679}$. Since 2017 DevOps and database “Domino Xpage database building automation or continuous integration using Jenkins with maven.” in $Q_{43092239}$. From mid-2011 to mid-2014, DB Stored Procedure topic remains dominant. “Oracle APEX: Call stored procedure from javascript” in $Q_{20501834}$.

Platform Adoption Category From 2008 to mid-2011, Platform Adoption related topics were the most dominant (e.g., “Suggested platform/tools for rapid game development and game prototyping” in Q_{312357}). Between mid-2011 to mid-2017, Authentication & Authorization topic becomes dominant (e.g., “Can I implement my own authentication process in force.com or it is against terms of service?” $Q_{13059568}$, $Q_{13034866}$). Since the end of 2017, Platform Infrastructure API remains the most dominant. So, practitioners ask queries like “VirtualBox VM changes MAC address after imported to Oracle Cloud Infrastructure” in $Q_{61501108}$ and “How to send a classic report as mail body in oracle Apex 19.2” in $Q_{59693984}$, report layout ($Q_{59833909}$, $Q_{59752159}$).

Platform Maintenance Topic Category From 2008 to mid-2019, the Build Configuration Management topic remains the most dominant topic. It has some high spikes in the number of questions during the beginning of 2012 and the first quartile of 2014. Build error $Q_{21720165}$, $Q_{21326163}$, build projects automatically $Q_{21758244}$. From mid-2019, Library Dependency Management topic-related questions became popular (e.g., library-related issues (e.g., $Q_{62825046}$, $Q_{61100705}$), library not found$Q_{61911916}$).

Third-Party Integration Topic Category. The five topics from this category evolve simultaneously. From the beginning of 2015, the External Web Request Processing topic has become more dominant than other topics with a slight margin. External Web Request Processing and Fetch & Process API response, E-signature topics become dominant during the pandemic with queries such as platform support on e-signature $Q_{62417381}$ and etc.

In Figure 3.8, we now provide more insight into the evolution of LCSD topic categories. It confirms the findings presented in Figure 3.7 and adds some previously unknown insights. For instance, in the last quartile of 2009, it is apparent that Data Storage is the most popular Topic Category. According to the absolute impact metric, all five themes are increasing monotonically. The relative impact measure, on the other hand, indicates that the Customization, Platform Maintenance, and Third-Party Integration Topic group evolves in a nearly identical manner. However, this Figure demonstrates that, beginning in 2016, Platform Adoption-related conversation increased and eventually surpassed Data Storage-related discussion. This in-depth examination of evolution is significant because it demonstrates that, while Data Storage Topics are the second-largest Topic category, Platform Adoption-related queries are evolving rapidly and require further attention from platform vendors.

RQ2. How does the LCSD-related discussion evolve? Since 2012, LCSD-related talks in SO have grown in popularity, and this trend has accelerated since 2020. Initially, the Customization and Data Storage Topic Categories dominated, but in recent years, Platform Adoption-related inquiries have grown in popularity.

3.4.3 What types of questions are asked across the observed topic categories? (RQ3)

3.4.3.1 Motivation

This research question aims to provide a deeper understanding of LCSD-related topics based on the types of questions asked about the LCSD platforms in SO. For example, "what" types of questions denote that developers are not sure about some specific characteristics of LCSD platforms, while "how" types of questions denote that they do not know how to solve a problem using an LCSD platform. Intuitively, the prevalence of "what" types of questions would denote that the LCSD platforms need to better inform the services they offer, while the prevalence of "how" type of questions would denote that the LCSD platforms need to have better documentation so that developers can learn easily on how to use those. Initially, in 2011 Treude et al. [241] investigated different types of questions on stack overflow. Later Rosen et al. [214] conducts an empirical study like ours on Mobile developers' discussion in stack overflow with these four types of questions. Later, very similar studies on chatbot development [2] and IoT developers' discussion on Stack overflow [254] also explore this research question to provide more insights about specific domains and complement the findings of topic modeling.

3.4.3.2 Approach

In order to understand what-type of questions are discussed in SO by the LCSD practitioners, we take a statistically significant sample from our extracted dataset and then manually analyze each question and label them into one of four types: How-type, Why-type, What-type, Others-type following related studies [2, 214, 254]. So, our approach is divided into two steps: Step 1. We generate a statistically significant sample size, Step 2. we manually analyze and label them.

Step 1. Generate Sample. As discussed in Section 4.3.1 our final dataset has 26,763 questions. A statistically significant sample with a 95% confidence level and five confidence intervals would be at least 379 random questions, and a 10 confidence interval would have a sample size of 96 questions. A random sample represents a representative for the entire dataset, and thus this could miss questions from the subset of questions that may belong to smaller topic categories. For example, as discussed in RQ1, we have 40 topics organized into five categories. As random sampling is not uniform across the topic categories, it might miss important questions from smaller topic categories such as Third-Party Integration. Therefore, following previous empirical studies [2, 254], we draw a statistically significant random sample from each of the five topic categories. Specifically, we draw the distribution of questions in our sample from each of the 5 topic categories with a 95% confidence level and ten confidence intervals. The sample is drawn as follows: 95 questions from the Customization category (total question 8014), 95 questions from the Data Storage category (total question 6610), 94 questions from Platform Adoption category (total question 5285), 94 questions from Platform Maintenance category (total question 3607), 93 questions from Third-Party Integration category (total question 3247). In summary, we sampled a total of 471 questions.

Step 2. Label Question Types. We analyze and label each question from our samples into the following four categories. The categories and the coding guides follow previous research [2, 215, 254]

- **How-type** post contains a discussion about the implementation details of a technical task [254]. The questions primarily focus on the steps required to solve certain issues or complete certain tasks (e.g., "How to create a submit button template in Oracle APEX?" in $Q_{1730566}$).

- **Why-type** post is about troubleshooting and attempting to determine the cause/reason for a behavior. These questions help practitioners understand the problem-solving or debugging approach, e.g., in $Q_{25176669}$, a user is trying to find out why an SSL server certificate is rejected.

- **What-type** question asks for more information about a particular architecture/event. The practitioners ask for

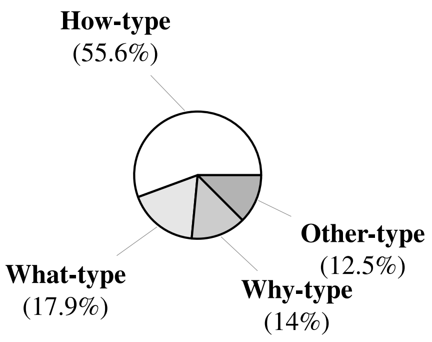


Figure (3.9) Distribution of different question Types

Table (3.1) Types of questions across the LCSD topic categories

Topic Category	**How**	**What**	**Why**	**Other**
Customization	51.0%	18.4%	17.3%	13.3%
Data Storage	59.8%	13.4%	17.5%	9.3%
Platform Adoption	57.3%	19.8%	9.4%	13.5%
Platform Maintenance	49.0%	20.4%	17.3%	13.3%
Third-Party Integration	61.2%	17.3%	8.2%	13.3%
Overall	55.6%	17.9%	14.0%	12.5%

more information that helps them to make informed decisions. For example, in $Q_{11608661}$ a practitioner is asking for detailed information about the Oracle Apex platform's secure cookies.

- **Other-type** question do not fall into any of the above three categories, e.g., "Initiating Salesforce API in Google App Script" in $Q_{66317111}$

Three authors (first, third and fourth) participated together in the labeling process. We assessed our level of agreement using Cohen kappa [169]. The disagreement and annotation difficulties were resolved by discussing with the first author. In general, the authors achieved a substantial agreement ($k > 0.6$) on the 471 questions classified. Our coding guidelines and the final annotated dataset are available in our replication package.

3.4.3.3 Result

Table 3.1 shows the percentage of type of questions across our five LCSD topic categories. Similar to related studies [2, 215, 254], during our labeling process, we observed that some of the questions can have multiple labels, e.g., What-type and How-type. For example, "How can I get jQuery post to work with a [platform] WebToLead" in $Q_{2339550}$ discusses making Ajax request using jQuery where the practitioner is getting an error response. At the same time, the practitioner is further querying some detailed information on how Ajax requests. Therefore, the sum of percentages of question types reported in the result section is more than 471 classified posts. We now discuss each question type with examples.

How-type. Around 57% of our annotated questions fall under this type. This type of question is most prevalent in the topic categories Third-Party Integration (61%) followed by Data Storage (60%), Platform Adoption (57%), Customization (51%), Platform Maintenance (49%). This high prevalence is not surprising, given that SO is a Q&A platform and the LCSD practitioners ask many questions about how to implement certain features or debug an issue. Additionally this also signifies that LCSD practitioners are asking a lot questions while integrating with third-party library (e.g., $Q_{62825046}$) and plugins (e.g., $Q_{61455233}$) and managing the data with a database management system (e.g., $Q_{38111768}$) or file storage (e.g., $Q_{63284305}$). To explain further we find questions regarding implementing encryption, e.g., $Q_{2220076}$, or Making HTTP POST request, e.g., $Q_{32736416}$, or debugging a script, e.g., $Q_{45619586}$, or implement a feature, e.g., $Q_{28990848}$ etc.

What-type. This is the second biggest question type with 18% of all annotated questions. This type of question is the most dominant in the topic categories Platform Maintenance (20.5%), Platform Adoption (20%), and Customization (18%). This type of question can be associated with How-type questions, where the practitioners require further information to implement certain features. For instance, in this question, a practitioner is querying about "How to implement circular cascade select-lists" in $Q_{60676786}$. The questions in this category signify that practitioners fail to find relevant information from the official documentation (e.g., $Q_{9377042}$) sources. Therefore, as this type of question is prevalent in Platform Maintenance and Platform Adoption category, LCSD platform providers might focus more on improving their resources. As an example, we find questions on JavaScript events not working correctly, e.g., $Q_{51564154}$, roll back changes, e.g., $Q_{11156810}$, designing workflow, e.g., $Q_{11156810}$.

Why-type. This is the third most prevalent question type category, with 14% of all annotated questions. This type of question is the most prevalent in the topic categories Customization, Data Storage, and Platform Maintenance with around 17% questions. These questions are mostly related to troubleshooting like when LCSD practitioners implement particular features or deploy an application. For instance, e.g., "Why does this error happen in [Platform]?" in $Q_{48818859}$, "Why isn't the document going into edit mode" in $Q_{51660117}$, "Not able to launch Android Hybrid app using [Platform] Mobile SDK" in $Q_{20417235}$, "Java code running twice" in $Q_{17147921}$.

Other-type. Around 14% of our annotated questions fall under this type. The questions are almost uniformly distributed across the five topic categories. The questions contain general problems, e.g., "UTF-8 character in attachment name" in $Q_{22808965}$ or "Domino Server is installed on Unix or Windows?" in $Q_{10796638}$. Some of the questions in this type also contain multiple/ambiguous questions (e.g., $Q_{27896327}$). For example, How to test an application?, which library is better? etc.

RQ3. What types of questions are asked across the observed topic categories? How-type (57%) questions are the most prevalent across all five topic categories, followed by What-type (18%), Why-type (14%), and Other-type (12%) questions. Practitioners in the Customization and Platform Maintenance topic categories are more interested in troubleshooting, i.e. (Why-type, What-type). Practitioners generally ask more implementation questions (i.e., How-type) in the Third-Party Integration Category. Practitioners in the Data Storage topic category are interested in designing databases (i.e., How-type) and troubleshooting (i.e., What-type, Why-type). This indicates the necessity for a more robust community for troubleshooting and debugging issues.

3.4.4 How are the observed topic categories discussed across SDLC phases? (RQ4)

3.4.4.1 Motivation

As we observed the prevalence and evolution of diverse LCSD topics in SO, we also find that the topics contain different types of questions. This diversity may indicate that the topics correspond to the different SDLC phases that are used to develop low code software development (see Section 5.3 for an overview of the LCSD phases). For example, intuitively What-type of questions may be due to the clarification of a low code software requirements during its design phases, which questions/topics related to troubleshooting of issues may be asked during the development, deployment, and maintenance phase. Therefore, an understanding of the SDLC phases in the LCSD questions in SO may offer us an idea about the prevalence of those SDLC phases across our observed LCSD topics in SO. This understanding may help the LCSD practitioners to determine how SO can be used during low code software development across the various SDLC phases.

3.4.4.2 Approach

In order to understand the distribution of LCSD topics across agile SDLC phases, we collect a statistically significant sample of questions from our extracted dataset D into one of the six Agile software development methodology [48] phases: (1) Requirement Analysis & Planning, (2) Application Design, (3) Implementation, (4) Testing, (5) Deployment, and (6) Maintenance. First, we generate a statistically significant sample size. We use the same set of randomly selected (i.e., 471) posts that we produced during RQ3 (see Section 3.4.3). So, we take a statistically significant stratified random sample for each topic category in our dataset with 95% confidence level and 10 confidence interval to ensure that we have a representative sample from each topic category [2, 254]. We manually annotate each question post with one/more SDLC phases.

We followed the same annotation strategy to label SDLC phased as we did for RQ3 (see Section 3.4.3.2). Each question was labeled by at least two authors (second and third/fourth) after extensive group discussion on formalizing annotation guidelines and participating in joint sessions. We find our level of agreement using Cohen kappa [10, 169].

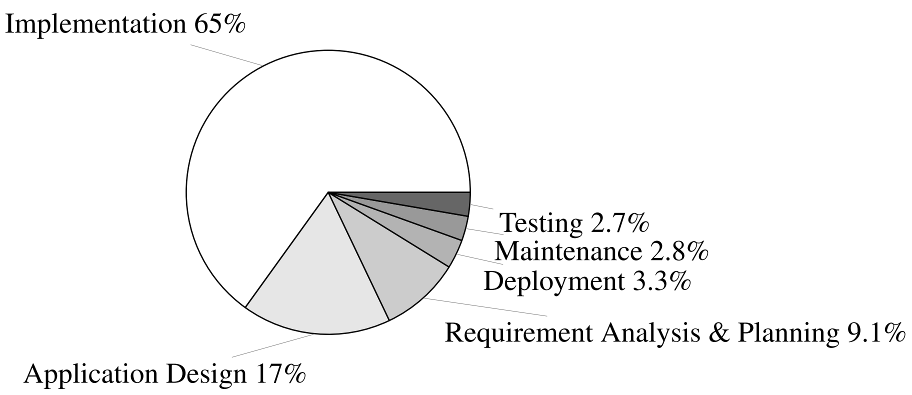


Figure (3.10) Distribution of questions (Q) per SDLC phase

The authors generally achieved a substantial agreement ($k > 0.70$). For example, a new practitioner is tasked with finding the right LCSD platform during the planning stage of his/her LCSD application. The practitioner queries, "Are there any serious pitfalls to Outsystems Agile Platform?" ($Q_{3016015}$). We thus assign the SDLC phase as "Requirement Analysis & Planning". Another question asks, "Google App Maker app not working after deploy" ($Q_{42506938}$). We label the SDLC phase as "Deployment". For some questions, it involved significant manual assessment to assign appropriate SDLC phase, e.g., Requirement Analysis & Planning phase vs Application Design and Application Design vs Implementation phase. As such, we developed a detailed annotation/coding guide to help us with the manual assessment. This annotation guide was constantly updated during our study to ensure that the guide remained useful with all relevant instructions. For example, one of the questions that helped refine our annotation guide is the question noted by the respected reviewer, i.e., "Can AppMaker be utilized with SQL Server?" in $Q_{55220499}$. The user in this question wants to know if Google App Maker and SQL Server databases can be connected. This question was categorized as Application design phase. Based on this, according to our annotation guideline, this question can be labelled as Requirement Analysis & Planning phase too. However, after discussion, the first and third authors agreed to label it as Application Design phase because from the problem description, it seems the question mainly focuses on connecting the application to a custom data storage. As this question focuses on data source design, which is often explored during the Application Design phase, we concluded that it should be labeled as such. The labeling of each question to determine the precise SDLC phases was conducted by several co-authors in joint discussion sessions spanning over 80 person-hours.

3.4.4.3 Results

Figure 4.6 shows the distribution of our LCSD questions into six agile SDLC phase. We find that the Implementation phase has 65% of our 471 annotated questions, followed by Application Design (17%), Requirement Analysis & Planning (9.1%). It is not surprising that the Implementation phase has so many questions because SO is a technical

Q&A platform and practitioners use it mostly to find issues when trying to implement some feature. Though the percentage of questions is not very high (e.g., between 2-3%), we also find practitioners ask questions regarding Testing and Deployment phases too (e.g., "Automated Testing for Oracle [Platform] Web Application" in $Q_{1764497}$). This analysis highlights that LCSD practitioners ask questions regarding the feasibility analysis of a feature to make design decisions to implement the feature to deployment. We provide an overview of the types of questions asked during these six SDLC phases.

Requirement Analysis & Planning (43, 9.1%). Requirement analysis is the first and most important stage of software development because the application largely depends on this. Requirement analysis is a process to develop software according to the users' needs. In agile software development methodology, features are implemented incrementally, and requirement and feasibility analysis are crucial in implementing a new feature. During this phase, the operational factors are considered, and the feasibility, time-frame, potential complexity, and reliability. Requirement management tools are typically included with LCSD systems, allowing developers to collect data, modify checklists, and import user stories into sprint plans. Throughout this stage, developers tend to ask questions regarding the platform's features (e.g., "Does Mendix generates a source code in any particular language, which can be edited and reused?" in $Q_{53043346}$), learning curve (e.g., $Q_{55304547}$, $Q_{45631057}$), and the LCSD platform's support for faster application development (e.g., $Q_{28983651}$), general deployment/maintenance support (e.g., $Q_{50460088}$) in order to select the best platform for their needs. For example, in this popular question, a new practitioner is asking for some drawbacks on some potential pitfalls for a particular LCSD platform, e.g., "Are there any serious pitfalls to [Platform] Agile Platform?" ($Q_{3016015}$). A developer from that platform provider suggests using the platform to build an application and decide for himself as it is hard to define what someone might consider a pitfall. In another question, a practitioner is asking if it is possible to integrate Selenium with an LCSD platform (e.g., $Q_{52010004}$)

Application Design (80, 17%). The design specification is created in this step based on the application's needs. The application architecture (e.g., $Q_{53820097}$), modularity, and extensibility are all reviewed and approved by all critical stakeholders. The LCSD developers face challenges regarding data storage design, drag and drop UI design, connecting on-premise data-sources with the LCSD platform (e.g., "Can AppMaker be used with SQL Server" ($Q_{55220499}$)), data migration to LCSD platform ($Q_{46421271}$), following best practices (e.g, "Salesforce Best Practice To Minimize Data Storage Size" in $Q_{14073151}$), designing a responsive web page (e.g., ($Q_{52744026}$)).

Implementation (306, 65%). The actual application development begins at this phase. LCSD developers confront a variety of obstacles when they try to customize the application (i.e., personalize UI (e.g, $Q_{6454308}$), implement business logic (e.g, $Q_{40472354}$)), integrate third-party plugins(e.g, $Q_{46538734}$), debug (e.g, $Q_{35898112}$) and test the implemented functionality. For example, LCSD practitioners ask customization questions such as How can they change the timezone in a platform in $Q_{47731051}$, customizing UI in $Q_{40159662}$. Many of these challenges arise from incomplete or incorrect documentation. In $Q_{34510911}$, an LCSD developer asks for sample code to convert a web page to a PDF. The

Table (3.2) Distribution (frequency) of LCSD topics per SDLC phase. Each colored bar denotes a phase (Black = Requirement Analysis, Green = Application Design, Magenta = Implementation, Red = Testing, Blue = Deployment, Orange = Maintenance)

Topics	Development Phases Found in #Questions
Customization (95)	5 17 71 1 1
Data Storage (95)	6 16 70 2 1
Platform Adoption (94)	17 21 51 1 4
Platform Maintenance (94)	11 9 46 10 12 6
Third-Party Integration (93)	4 17 68 1 2 1

official documentation is not sufficient enough for entry-level practitioners.

Testing (13, 2.7%). LCSD testing differs from standard software testing in some fundamental ways. In LCSD development, many of the features are implemented using graphical interfaces, and they are provided and tested by the LCSD platform providers. As a result, unit testing is less important compared to traditional software development. In LCSD approach practitioners face difficulties to lack of documentation of testing approach in LCSD platform (e.g, "How to bypass login for unit-testing [Platform]?" in $Q_{54432666}$), test coverage (e.g, $Q_{54899980}$, $Q_{57755398}$), automated testing (e.g, "[Platform] 20.1 automated testing" $Q_{63594106}$), testing browser compatibility (e.g, Q), troubleshooting errors while running tests (e.g, $Q_{47254010}$) etc.

Deployment (16, 3.3%). At this phase, the feature of the application needs to be deployed for the targeted users. One of the goals of LCSD development is to handle many of the complexities of the *deployment and maintenance* phase. Many LCSD platform providers provide advanced Application Life-Cycle Management tools to deploy and maintain the staging (i.e., testing) and the production server (e.g., $Q_{65124133}$). However, LCSD practitioners still face many challenges regarding deployment configuration issues ($Q_{46369742}$), Domain name configuration (e.g., DNS configuration (e.g, $Q_{65678735}$), SSL Configuration (e.g, $Q_{67186273}$)), accessibility issues such as with public URL ($Q_{44136328}$, $Q_{53884162}$)) etc. For example, in this post, a practitioner is having deployment issues (e.g., "[Platform] app not working after deployment" ($Q_{42506938}$)). A community member provides a detailed description of how to accomplish this in the answer, highlighting the lack of *Official Documentation* for such a critical use-case. There are a few questions concerning delivering an app with a custom URL or domain name (for example, "How to make friendly custom URL for deployed app" in $Q_{47194231}$). It was challenging in this scenario because the platform did not have native support.

Maintenance (13, 2.8%). At this phase, the LCSD application is deployed and requires ongoing maintenance. Sometimes new software development life cycle is agile (i.e., incremental) because new issues are reported that were previously undiscovered and request new features from the users. LCSD practitioners face some problems at this phase, such as event monitoring (e.g, $Q_{64322219}$), collaboration and developers role management (e.g, "Role based hierarchy in report access" in $Q_{10436719}$ or $Q_{52762374}$), and application reuse (e.g, $Q_{64276891}$), application version, i.e., "Do I have the latest version of an [Platform] component?" in $Q_{45209796}$ or $Q_{52762374}$, etc.

Table (3.3) Types of questions across the Software development life cycle phases

SDLC phase	How	What	Why	Other
Requirement Analysis & Planning(9%)	28%	35%	7%	30%
Application Design(17%)	75%	16%	4%	10%
Implementation(65%)	59%	17%	16%	11%
Testing(3%)	62%	15%	15%	8%
Deployment(3%)	31%	25%	38%	12%
Maintenance(3%)	46%	15%	31%	15%

Topic Categories in different SDLC phases. We find that for all five topic categories, LCSD practitioners need some community support from planning to debugging to deployment (e.g., "How does one deploy after building on [platform]" in $Q_{3952481}$). We report how LCSD topics and different types of questions are distributed across six SDLC phases. Table 4.1 shows the distribution of SDLC phases for each topic category. Our analysis shows that for the Customization topic Category, most questions are asked during the Implementation (75%) and Design (18%) phases. The most dominant SDLC phase, i.e., the Implementation phase, is most prevalent in Customization (75%), Data Storage (74%), and Third-Party Integration (73%). Requirement Analysis phase is dominant in Platform Adoption (18%) and Platform Maintenance (12%) topic categories where practitioners ask questions like "Disadvantages of the [platform]" in $Q_{1664503}$. Similarly, question in Platform Maintenance topic category is also prevalent in Testing (11%), deployment (13%), and Maintenance (6%) SDLC stage.

Types of questions in different SDLC phases. We report the distribution of question types across SDLC phases in Table 3.3. It shows that for Requirement Analysis & Planning phase, most questions (35%) belong to What-type. This insight signifies that at this phase, practitioners are making inquiries about feature details (e.g., $Q_{9577099}$). In the Application Design, Implementation, and testing phase, most of the questions belong to How-type, i.e., practitioners are querying about how they can implement a particular feature (e.g., $Q_{13933003}$) or test it (e.g., $Q_{9594709}$). At the Deployment phase most prominent is Why-type (38%) followed by How-type(31%). We can see a similar pattern for the Maintenance phase, where the most significant question type is How-type (46%) followed by Why-type (31%). We see this pattern because, at the Deployment and Maintenance phase, most of the questions belong to some server configuration error (e.g., $Q_{4497228}$) and the practitioners' inquiry about how they can set up specific server settings (e.g., $Q_{8148247}$). Similarly, we find that What-type questions are more prevalent during Requirement Analysis and Deployment phases.

RQ4. How are the observed topic categories discussed across SDLC phases? Among six agile SDLC phases, the Implementation phase is the most prevalent (65% questions), followed by Application Design (17%), Requirement Analysis & Planning (9.1%), Deployment (3.3%), Maintenance (2.8%) and Testing (2.7%). The Implementation Phase is most prevalent in all of the five topic categories and four question types. During Requirement Analysis, Testing, and Deployment phases, Platform Adoption and Platform Maintenance topic categories are more dominant. The How-type question is most popular in the Application Design phase, the what-type question is prevalent in the Requirement Analysis and Planning phase, and the why-type question is prevalent in the Deployment and Requirement Analysis phases.

3.4.5 What LCSD topics are the most difficult to get an accepted answer? (RQ5)

3.4.5.1 Motivation

After reviewing LCSD-related topics and discussions in the agile SDLC stages, we discovered that LCSD practitioners encounter generic software development problems and particular challenges specific to LCSD platforms (e.g., Platform Adoption, Platform Maintenance). Some posts come up repeatedly, and some have a lot of community participation (i.e., answers, comments, up-votes). As a result, not all topics and SDLC phases are equally difficult to get a solution. A thorough examination of the complexity and popularity of the practitioners' conversation might yield valuable information about how to prioritize research and community support. For example, LCSD platform providers and academics can take the required measures to make the architecture, design, features, and tools of LCSD platforms more useable for practitioners, particularly newbies.

3.4.5.2 Approach

We compute the difficulty of getting an accepted answer for a group of questions using two metrics for each question in that group (1) Percentage of questions without an accepted answer, (2) Average median time needed to get an accepted answer. In the same way, we use the following three popularity metrics to calculate popularity of that topic in the SO community: (1) Average number of views, (2) Average number of favorites (i.e., for each question number of users marked as favorite), (3) Average score.

The five metrics are standard features of a SO question, and many other related studies [2, 7, 10, 37, 254] have used them to analyze the popularity and difficulty of getting a solution for a question. In SO, one question can have multiple answers, and The user who posted the question has the option of marking it as accepted. Hence, the accepted answer is considered correct or sound quality. So, the absence of an accepted answer may indicate the user did not find a helpful, appropriate answer. The quality of the question (i.e., problem description) might be one reason for not getting an acceptable answer. However, the SO community collaboratively edits and improves the posts. Therefore, the lack of an accepted answer most likely indicates that the SO community finds those questions challenging to answer. The success and usefulness of a crowd-sourced platform such as SO depends on the community members to quickly provide relevant, helpful correct information. In SO, the median time to get an answer is around 21 minutes only [254], but a complicated or domain-specific question may necessitate additional time to receive an accepted answer.

It can be non-trivial to assess the popularity and difficulty of getting an accepted answer for the topics using multiple metrics. We thus compute two fused metrics following related works [254]. We describe the two fused metrics below.

Fused Popularity Metrics. First, we compute the popularity metrics for each of the 40 LCSD topics. However, the average view counts can be in the range of hundreds, average scores, and average favorite count between 0-3. Therefore, following related study [254] we normalize the values of the metrics by dividing the metrics by the average of the metric values of all the groups (e.g., for topics $K = 40$). Thus, we create three new normalized popularity metrics for each topic. For example the normalized metrics for a group i for all the K groups can be $ViewN_i$, $FavoriteN_i$, $ScoreN_i$ (e.g., for LCSD topics $K = 40$). Finally, We calculate the fused popularity $FusedP_i$ of a group i by taking the average of the three normalized metric values.

$$ViewN_i = \frac{View_i}{\frac{\sum_{j=1}^{K} View_j}{K}} \tag{3.5}$$

$$FavoriteN_i = \frac{Favorite_i}{\frac{\sum_{j=1}^{K} Favorite_j}{K}} \tag{3.6}$$

$$ScoreN_i = \frac{Score_i}{\frac{\sum_{j=1}^{K} Score_j}{K}} \tag{3.7}$$

$$FusedP_i = \frac{ViewN_i + FavoriteN_i + ScoreN_i}{3} \tag{3.8}$$

Fused Difficulty Metrics. Similar to popularity metrics, we first compute the difficulty metrics for each topic. Then we normalize the metric values by dividing them by the average of the metric value across all groups (e.g., 40 for LCSD topics). Thus we, create two new normalized metrics for a given topic i. Finally, We calculate the fused difficulty metric $FusedD_i$ of topic i by taking the average of the normalized metric values.

$$PctQuesWOAccAnsN_i = \frac{PctQWoAcceptedAnswer_i}{\frac{\sum_{j=1}^{K} PctQWoAcceptedAnswer_j}{K}} \tag{3.9}$$

$$MedHrsToGetAccAnsN_i = \frac{MedHrsToGetAccAns_i}{\frac{\sum_{j=1}^{K} MedHrsToGetAccAns_j}{K}} \tag{3.10}$$

$$FusedD_i = \frac{PctQuesWOAccAnsN_i + MedHrsToGetAccAnsN_i}{2} \tag{3.11}$$

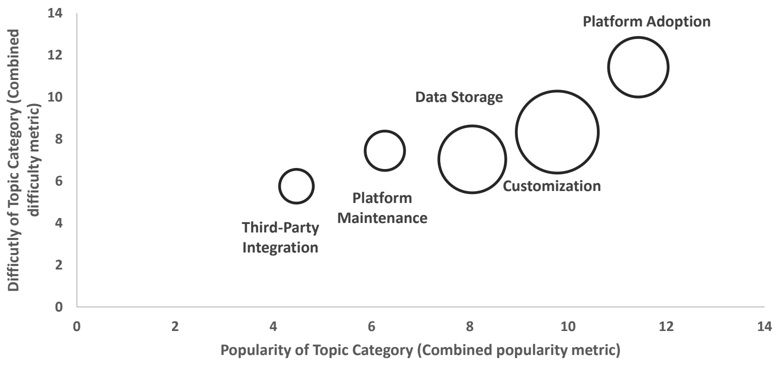


Figure (3.11) The popularity vs. difficulty of getting an accepted answer for LCSD Topic categories.

In addition to this, we also aim to determine the correlation between the difficulty and the popularity of the topics. We use the Kendall Tau correlation measure [141] to find the correlation between topic popularity and topic difficulty. Unlike Mann-Whitney correlation [148], it is not susceptible to outliers in the data. We can not provide the evolution of popularity and difficulty for these topics because SO does not provide the data across a time series for all metrics such as view count, score, etc. However, asLCSD-related topics are showing increasing trends in recent times, our analysis is valid for recent times.

3.4.5.3 Results

In Figure 4.7 we present an overview of the five high-level topic categories and their popularity and difficulty to get an accepted answer. In the Figure, the bubble size represents the number of questions in that category. The Figure shows that Platform Adoption is the most popular and challenging topic category to get an accepted answer, followed by Customization, Data Storage, Platform Maintenance, and Third-Party Integration. We can also see that three topic categories, Platform Maintenance, Data Storage, and Customization, are almost similar in terms of difficulty to get a solution. From our analysis, we find that practitioners find the Third-Party Integration topic category relatively less difficult because many questions in this category are also relevant to traditional software development (e.g., integrating Google Maps in $Q_{63457325}$ and $Q_{1258834}$) and thus easier to get community support. Similarly, we find that questions in the Platform Adoption topic category are quite specific to particular LCSD platforms and thus sometimes have less community support to find an acceptable answer quickly.

Topic Popularity. For each of the 40 topics, Table 4.2 shows three popularity metrics: Average number of 1. Views, 2. Favorites, 3. Scores. It also contains the combined popularity metrics (i.e., FusedP) that are based on the above three metrics and using the Equation 4.4. In the Table, the topics are presented in descending order based on the FusedP popularity metric.

Platform Related Query topic from the Platform Adoption Category has the highest FusedP score. It also has the

Table (3.4) Popularity for getting an accepted answer for LCSD topics

Topic	Category	FusedP	#View	#Favorite	#Score
Platform Related Query	Platform Adoption	3.45	2229.9	0.9	2.6
Message Queue	Platform Adoption	1.49	1671.7	0.3	1.1
Dynamic Page Layout	Customization	1.31	2447.2	0.2	0.7
Build Config. Management	Platform Maintenance	1.22	1522.1	0.2	1
Pattern Matching	Customization	1.17	1539.5	0.2	0.9
SQL CRUD	Data Storage	1.15	1615.7	0.2	0.8
Web-Service Communication	Platform Adoption	1.15	1454.9	0.2	0.9
Misc. SWE Discussion	Platform Adoption	1.13	1193.4	0.2	1
Interactive Report	Platform Adoption	1.12	1680.8	0.2	0.7
Fetch & Process API Response	Third-Party Integration	1.11	1277.8	0.2	0.9
Data Security & Replication	Data Storage	1.1	1411.7	0.2	0.8
Hosting Config. & SEO	Platform Maintenance	1.09	1377.9	0.2	0.8
Asynchronous Batch Jobs	Platform Maintenance	1.06	1245.7	0.2	0.8
Authentication & Authorization	Platform Adoption	1.05	1057.6	0.2	0.9
Library Dependency Mngmt	Platform Maintenance	1.05	1230.6	0.2	0.8
Email Processing	Third-Party Integration	1.05	1600	0.2	0.6
Formatted Data Parsing	Customization	1	1607.8	0.1	0.9
File Management	Data Storage	0.98	1302.6	0.2	0.6
DB Setup & Migration	Data Storage	0.97	1282.9	0.2	0.6
Testing	Platform Maintenance	0.96	1810.5	0.1	0.7
Date & Time Manipulation	Customization	0.94	1720.5	0.1	0.7
DB Stored Procedure	Data Storage	0.94	1715.5	0.1	0.7
Dynamic Data Binding	Customization	0.92	1810.5	0.1	0.6
External Web Req Processing	Third-Party Integration	0.91	831.8	0.2	0.7
App Deployment	Platform Maintenance	0.89	757.3	0.2	0.7
Semi-Structured Data Proc.	Data Storage	0.88	1096.5	0.2	0.5
Upgradation & Compatibility	Third-Party Integration	0.88	559.1	0.2	0.8
Dialog Box Manipulation	Customization	0.79	1479.4	0.1	0.5
Dynamic Event Handling	Customization	0.77	1245.1	0.1	0.6
Dynamic Form Controller	Customization	0.76	1382.1	0.1	0.5
Conditional BPMN	Customization	0.75	1335.1	0.1	0.5
SQL Syntax Error	Data Storage	0.75	1343	0.1	0.5
Graph/Chart	Platform Adoption	0.75	1156.3	0.1	0.6
User Role Management	Platform Adoption	0.73	1058.3	0.1	0.6
Window Style Manipulation	Customization	0.71	1000	0.1	0.6
Entity Relationship Mgmt	Data Storage	0.71	1178	0.1	0.5
Dynamic Data Filtering	Customization	0.66	1148.1	0.1	0.4
Date-based filtering	Data Storage	0.57	781.9	0.1	0.4
Platform Infrastructure API	Platform Adoption	0.56	577.2	0.1	0.5
eSignature	Third-Party Integration	0.52	574.9	0.1	0.4

highest average favorite count (e.g., 0.90) and highest average score (e.g., 2.60). 1.7% of total questions. This topic contains discussion about LCSD platforms features of different platforms, software development methodologies such as Agile and RAD development. The topic Message Queue under Platform Adoption category has the second highest FusedP value. This topic is about different asynchronous service-to-service data exchange mechanisms such as using a message queue. It generally contains discussions about popular micro-service design patterns. The topic Dynamic Page Layout under Customization categories is the third most popular topic and it has the highest average view count

Table (3.5) Difficulty for getting an accepted answer for LCSD topics

Topic	**Category**	**FusedD**	**Med. Hrs To Acc.**	**Ques. W/O Acc.**
Message Queue	Platform Adoption	1.86	21.4	61
Library Dependency Mngmt	Platform Maintenance	1.78	18.8	70
Web-Service Communication	Platform Adoption	1.76	19.8	60
Authentication & Authorization	Platform Adoption	1.67	17.4	68
Platform Infrastructure API	Platform Adoption	1.62	16.3	70
Fetch & Process API Response	Third-Party Integration	1.6	16.8	64
External Web Req Processing	Third-Party Integration	1.37	13.1	64
App Deployment	Platform Maintenance	1.35	12.1	70
Hosting Config. & SEO	Platform Maintenance	1.35	12.6	65
eSignature	Third-Party Integration	1.32	11.4	71
File Management	Data Storage	1.24	11.4	62
Asynchronous Batch Jobs	Platform Maintenance	1.19	10.8	60
Dynamic Page Layout	Customization	1.15	10.1	61
User Role Management	Platform Adoption	1.03	8.3	60
Graph/Chart	Platform Adoption	0.99	6.6	68
Platform Related Query	Platform Adoption	0.99	8.4	54
DB Stored Procedure	Data Storage	0.97	7.7	57
DB Setup & Migration	Data Storage	0.95	6.9	60
Dynamic Form Controller	Customization	0.92	6.3	61
Testing	Platform Maintenance	0.92	6.7	59
Conditional BPMN	Customization	0.85	5.7	58
Build Config. Management	Platform Maintenance	0.85	6.4	53
Dynamic Event Handling	Customization	0.84	4.2	68
Semi-Structured Data Proc.	Data Storage	0.82	4.6	63
Email Processing	Third-Party Integration	0.8	4.7	59
Dialog Box Manipulation	Customization	0.79	4.8	57
Interactive Report	Platform Adoption	0.79	5	56
Dynamic Data Binding	Customization	0.77	5.1	53
Dynamic Data Filtering	Customization	0.71	4.8	48
Misc. SWE Discussion	Platform Adoption	0.71	4.8	48
Data Security & Replication	Data Storage	0.66	3.5	52
Upgradation & Compatibility	Third-Party Integration	0.66	4.1	48
Date-based filtering	Data Storage	0.65	2	61
SQL Syntax Error	Data Storage	0.62	1.7	60
Entity Relationship Mgmt	Data Storage	0.62	2.2	57
Formatted Data Parsing	Customization	0.61	3.1	49
Date & Time Manipulation	Customization	0.59	2.5	51
Window Style Manipulation	Customization	0.58	2.3	51
Pattern Matching	Customization	0.52	1.8	48
SQL CRUD	Data Storage	0.50	1.7	46

Table (3.6) Correlation between the topic popularity and difficulty

coefficient/p-value	**View**	**Favorites**	**Score**
% Ques. w/o acc. ans	-0.33/0.01	0.02/0.88	-0.17/0.15
Med. Hrs to acc. ans	-0.05/0.65	0.30/0.02	0.22/0.05

(e.g., 2447.2). The posts under this topic discuss about UI (i.e. page) customization, hiding or moving elements based on some user action or an event (e.g., disable a button for dynamic action in $Q_{8640964}$. The eSignature topic from Third-Party Integration is the least popular with only 1.15% of total questions, a fused value of 0.52. It has the lowest favorite and score count. This contains discussion about different issues and customization for electronic signature of documents, i.e., docusign about collecting user's agreement/permission for sales or account opening. This topic is not that much popular and easy to get an accepted answer because this requirement is not generalized and not all the low-code application requires this.

Topic Difficulty. In Table 4.3 we present the two difficulty metrics: for all the questions in a topic 1. Percentage of questions without accepted answers, 2. Median hours to get accepted answer. Similar to topic popularity, we also report the combined topic difficulty metrics (e.g., FusedD) using the Equation 4.7 and the above two difficulty metrics. The topics in Table 4.3 are presented in descending order based on the FusedD value.

Topic Message Queue under Platform Adoption category is the most difficult topic to get an accepted answer in terms of FusedD value. Most median hours to get accepted answers (21). This topic contains discussion about general micro-service architecture (i.e., producer and consumer) and well as LCSD platform-specific support for these architectures. This is why this topic is also second most popular topic. Library Dependency Mngmt topic from Platform Maintenance is the second most difficult topic to get an accepted answer. Around 70% of its questions do not have any accepted answers. This topic concerns different troubleshooting issues about library and decencies of the system, server configuration, different library version compatibility issues. Web-Service Communication topic from Platform Adoption is the third most difficult topic. It has a long median wait time (around 20 hours) to get an accepted answer. This topic contains discussions about service-to-service communication via web service description language, HTTP REST message, and Windows Communication Foundation.

The topics that contain discussion about general software development (not specific to LCSD platforms) are the least difficult topics to get an accepted answer. For example, topic SQL CRUD under Data Storage category is the least difficult topic in terms of FusedD value (e.g., 0.5). This contains database CRUD related queries, and advanced queries too, such as inner join, nested join, aggregate. This also contains discussion about Object query language, which is a high-level wrapper over SQL. Topic SQL CRUD and SQL Syntax Error from the Data Storage category are two of the least difficult topics in terms of median hours to get accepted answers. Topic Pattern Matching and SQL CRUD are two of the least difficult topics in terms of questions without accepted answers.

Alternatively, topics that are specific to LCSD platforms are the most difficult topics. Four out of five most difficult topic belongs to Platform Adoption Categories. These questions can be popular as well as difficult. For example, LCSD-related Third-Party Integration related topic eSignature is the least popular topic from Table 4.2, is the most difficult topic in terms of questions without accepted answers (71%). Topic Platform Related Query is in the mid-range in terms of difficulty but most popular to get an accepted answer.

Correlation between Topic Difficulty and Popularity. Here we want to explore if there is any positive or negative relationship between topic popularity and difficulty. For example, Message Queue is the most difficult and, at the same time second most popular topic to get an accepted answer in terms of FusedD and FusedP metrics. Platform Related Query is the most popular but mid-range difficult topic.

Table 4.4 shows six correlation measures between topic difficulty and popularity in Table 4.2 and 4.3. Three out of six correlation coefficients are negative, and the other three are positive and they are not statistically significant with a 95% confidence level. Therefore, we can not say the most popular topic is the least difficult to get an accepted answer and vice versa. Nonetheless, LCSD platform provides could use this insight to take necessary steps. Most popular topics should have an easy to access-able answer (i.e., least difficult).

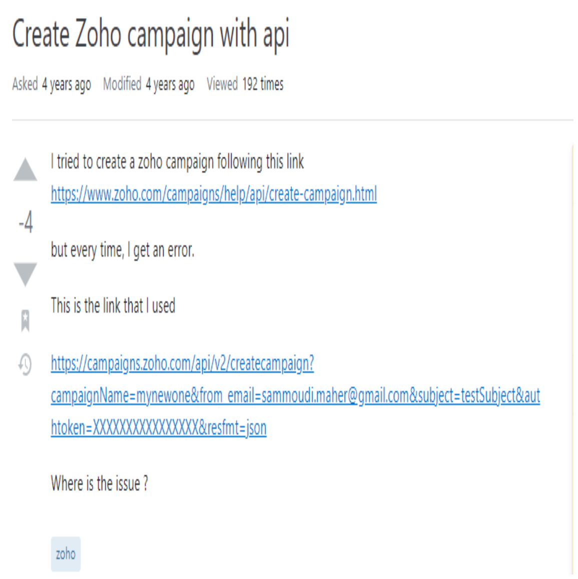
RQ5. What LCSD topics are the most difficult to answer? Platform Adoption is the most popular and challenging topic category, followed by Customization, Data Storage, Platform Maintenance, and Third-Party Integration. We also find that LCSD practitioners find Software Deployment and Maintenance phase most popular and difficult and Testing phase to be least difficult to an accepted answer. This indicates that LCSD platform providers should provide additional support to enable low-code practitioners understand and utilize the platform's features.

3.5 Discussions

During our analysis, we observed that several LCSD platforms are more popular across the topics than other platforms. We analyze our findings of LCSD topics across the top 10 most prevalent LCSD platforms in the dataset (Section 3.5.4). Finally, we discuss the implications of our study findings in Section 4.6.

3.5.1 Issues with not accepted answers or posts with negative score

In this chapter, for topic modeling we used questions and accepted answers only. We did not consider the posts with negative score too because of the following observations. (1) Many other similar empirical studies on Topic modeling on SO posts also considered the questions and accepted answers only, e.g., IoT developers discussions in SO [254], big data related discussion [37], concurrency related topics [7], mobile app development [214]. (2) A significant number of studies [23, 201, 209, 275] report quality of questions and unaccepted answers in SO are questionable and therefore it is quite a standard practice for SE researchers to consider the accepted answers in SO only. For example, In $Q_{7504057}$ (Fig 3.12) a user asks question about python code/package to connect and retrieve data from Salesforce. The accepted answer $A_{7504244}$ provides a relevant python code snippet the unaccepted answer $A_{34055640}$ provide resource link for a command line tool which may be relevant but exactly not what the user asked for. (3) Negative scored questions are typically incorrectly tagged (e.g., $Q_{4862071}$, $Q_{21377026}$, $Q_{37371712}$), duplicates (e.g., $Q_{12282151}$, $Q_{48121405}$), lack a detailed problem description (e.g., $Q_{25691340}$, $Q_{1974480}$, $Q_{50666660}$), lack correct formatting (e.g., $Q_{32208310}$). For instance, in $Q_{51635004}$ (Fig 3.12) a user inquires about an error encountered when attempting to contact a Zoho API. However, crucial important information such as an issue code or error message is lacking from the question description. In $Q_{4862071}$, an inexperienced user inadvertently tagged a question about the Oracle Apex platform with the Salesforce tag. We, therefore, choose not to include questions with a negative score or unaccepted answers. We also provide potentially missing out some insights for this choice in the threats to validity section (Section 4.7).



Create Zoho campaign with api

Asked 4 years ago Modified 4 years ago Viewed 192 times

-4

I tried to create a zoho campaign following this link

https://www.zoho.com/campaigns/help/api/create-campaign.html

but every time, I get an error.

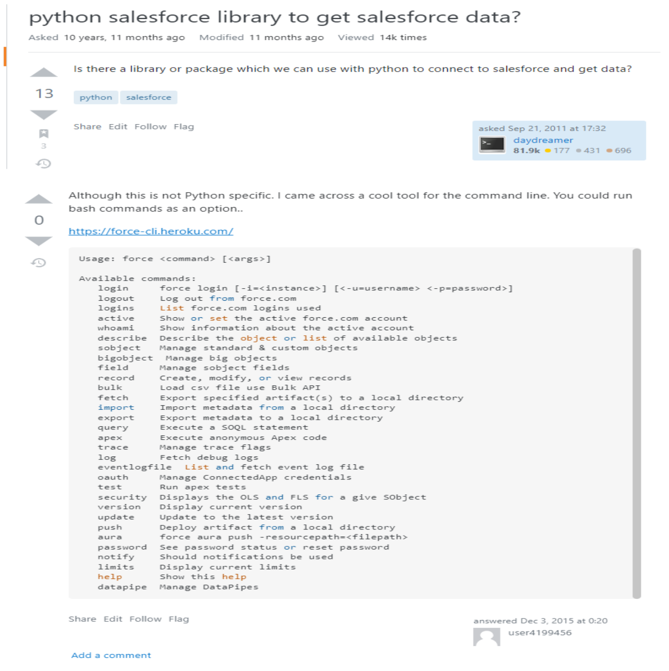
This is the link that I used

https://campaigns.zoho.com/api/v2/createcampaign?campaignName=mynewone&from_email=sammoudi.maher@gmail.com&subject=testSubject&authtoken=XXXXXXXXXXXXXXXXXX&resfmt=json

Where is the issue ?

zoho

(a) A question with negative score



python salesforce library to get salesforce data?

Asked 10 years, 11 months ago Modified 11 months ago Viewed 14k times

13

Is there a library or package which we can use with python to connect to salesforce and get data?

python salesforce

Share Edit Follow Flag

asked Sep 21, 2011 at 17:32 daydreamer 81.9k ● 177 ● 431 ● 696

0

Although this is not Python specific. I came across a cool tool for the command line. You could run bash commands as an option..

https://force-cli.heroku.com/

```
Usage: force <command> [<args>]

Available commands:
   login      force login [-i=<instance>] [<-u=username> <-p=password>]
   logout     Log out from force.com
   logins     List force.com logins used
   active     Show or set the active force.com account
   whoami     Show information about the active account
   describe   Describe the object or list of available objects
   sobject    Manage standard & custom objects
   bigobject   Manage big objects
   field      Manage sobject fields
   record     Create, modify, or view records
   bulk       Load csv file use Bulk API
   fetch      Export specified artifact(s) to a local directory
   import     Import metadata from a local directory
   export     Export metadata to a local directory
   query      Execute a SOQL statement
   apex       Execute anonymous Apex code
   trace      Manage trace flags
   log        Fetch debug logs
   eventlogfile  List and fetch event log file
   oauth      Manage ConnectedApp credentials
   test       Run apex tests
   security   Displays the OLS and FLS for a give SObject
   version    Display current version
   update     Update to the latest version
   push       Deploy artifact from a local directory
   aura       force aura push -resourcepath=<filepath>
   password   See password status or reset password
   notify     Should notifications be used
   limits     Display current limits
   help       Show this help
   datapipe   Manage DataPipes
```

Share Edit Follow Flag

answered Dec 3, 2015 at 0:20 user4199456

Add a comment

(b) A question with irreverent unaccepted answer

Figure (3.12) SO questions with a negative score or unaccepted SO answer.

3.5.2 Discontinued low-code platforms and future trends

From our analysis on Section 3.4.2 we see the evolution of LCSD platforms, especially from 2012. According to our data, we can see the discontinuation of some low-code platforms but they are usually soon replaced by new low-code/no-code services. For example, In Jan 2020, Google announced the discontinuation of Google App Maker [113] by 2021 [116]. But, shortly thereafter, Google announced a "no-code" platform called "AppSheet" [112] and promoted their fully managed serverless platform called AppEngine [19] to create web application promoting low-code approach. Microsoft and Amazon are also competing for superior low-code/no-code platforms with the emergence of new low-code service platforms such as Microsoft Power FX [174], Amazon Honeycode [128], AWS Amplify Studio [17]. The low-code approach is attracting increasing interest from traditional businesses, particularly during the pandemic [190].

3.5.3 LDA parameter Analysis

In this study, we applied LDA topic modelling, which employs Dirichlet distribution, to identify practitioners' discussions on low-code. As described in details in Section 4.3.2, we followed the industry standard to configure the parameters and hyperparameters and also followed the industry recommendation to manually annotate the topics as described in Section 3.4.1 in order to avoid sub-optimal solutions [75]. Following similar studies [2, 124, 254] we use the use the coherence score of of each model for different values of K. However, since LDA itself is probabilistic in nature [6] and can produce different results different runs on the same low-code dataset. In order to mintage this problem, we run our LDA model three times and compare the optimal number of topics. Fig. 3.13 shows the result of different coherence score for different values of K. Moreover, we can see after reaching highest coherence values for $K = 45$ the overall coherence score decreases as the value of K increases.

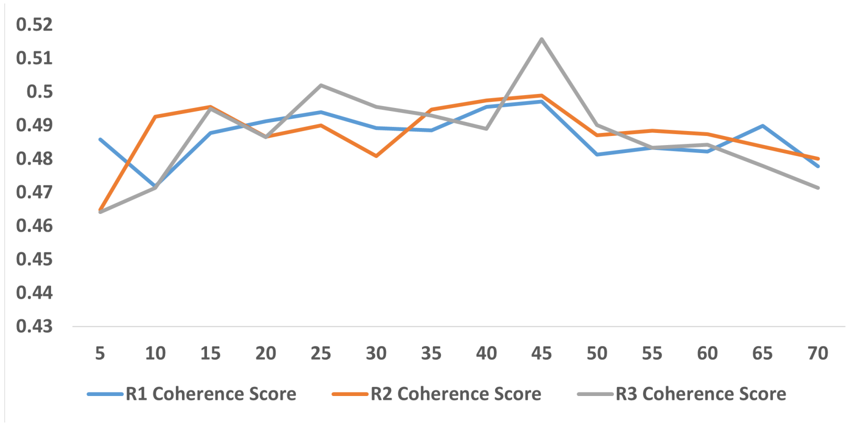


Figure (3.13) The different coherence values for varied K on different runs.

3.5.4 The Prevalence & Evolution of Top Ten LCSD platforms

Our analysis of the evolution of topic categories (see Section 3.4.2) shows that there is an overall increase in the number of new questions across the topics in SO. Our SO dataset is created by taking into account the LCSD platforms. In

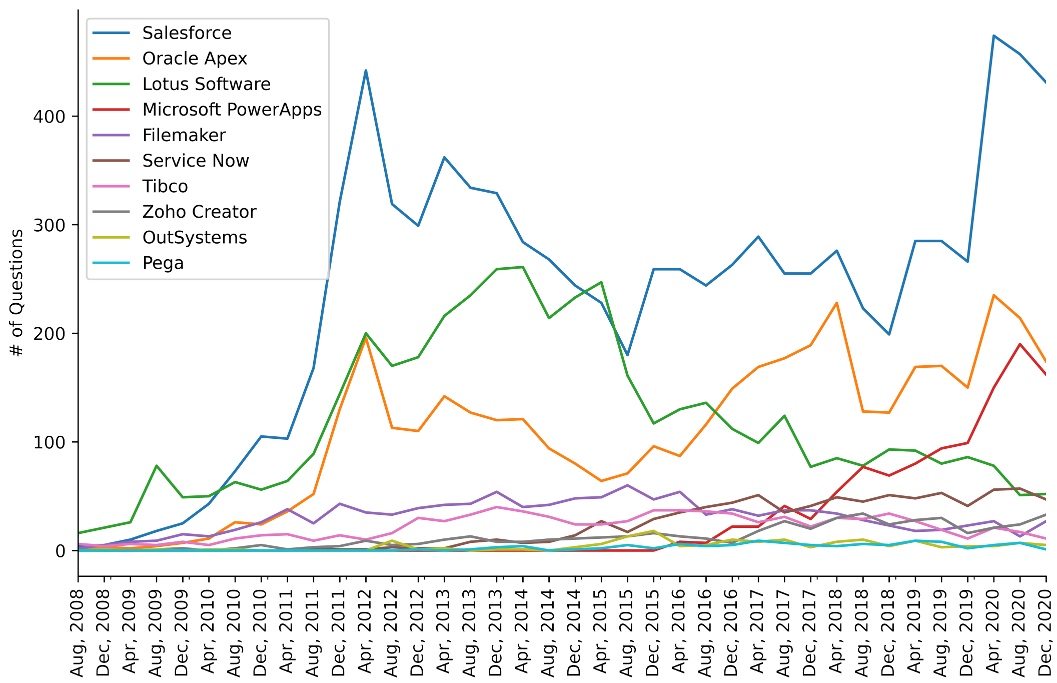


Figure (3.14) The evolution of top ten LCSD platforms discussions over time.

Figure 3.14, we show how the 10 LCSD platforms evolve in our SO dataset over the past decade based on the number of new questions. Salesforce [218] is the biggest and one of the oldest LCSD platforms (released in 1999) in our dataset with around 30% of all questions followed by Lotus Software [163], Oracle Apex [18], Microsoft powerapps [203]. Among these platforms, IBM Lotus Software was quite popular during the 2014s and gradually lost its popularity, and IBM finally sold it in 2018. Salesforce platform has been the most popular platform in terms of SO discussions since 2012. Our graph shows that these other three platforms, especially Microsoft Powerapps, are gaining lots of attention during the pandemic, i.e., early 2020.

We provide more context for these platforms in Figure 3.15 by illustrating the distribution of our observed five topic categories across the top ten LCSD platforms. We can see that Powerapps have the most number of queries in the Customization and Platform Adoption category. This happens because Powerapps is a relatively new LCSD platform (released in 2016) and it is gaining more and more attention from the community, and thus there are more queries such as business logic implementation (i.e., $Q_{61685582}$), connect Powerapps to a database in $Q_{61611950}$, user permission (i.e., $Q_{61838119}$). We can also see that older platforms such as Salesforce and Oracle Apex have more queries regarding Platform Maintenance, Third-Party Integration topic category. Practitioners ask many different questions regarding these platforms such as deployment-related (e.g., “Deploying a salesforce.com flex app without visualforce” in $Q_{6614226}$), third-party API integration (e.g., “Google Map Integrated with Salesforce is Blank” in $Q_{9028682}$), maintenance deployment (e.g., “Salesforce deployment error because of test class failure” in $Q_{9171945}$), interactive report in $Q_{9700660}$, customization with JSON $Q_{9833992}$, “what is dashboard?” in $Q_{10911269}$ and Oracle Apex how to use platform

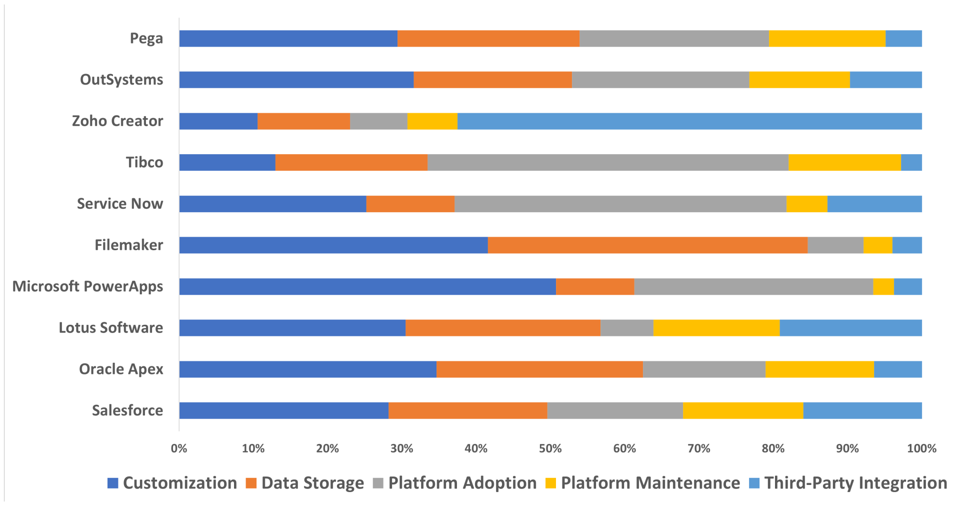


Figure (3.15) The distribution of topic categories across top ten LCSD platforms.

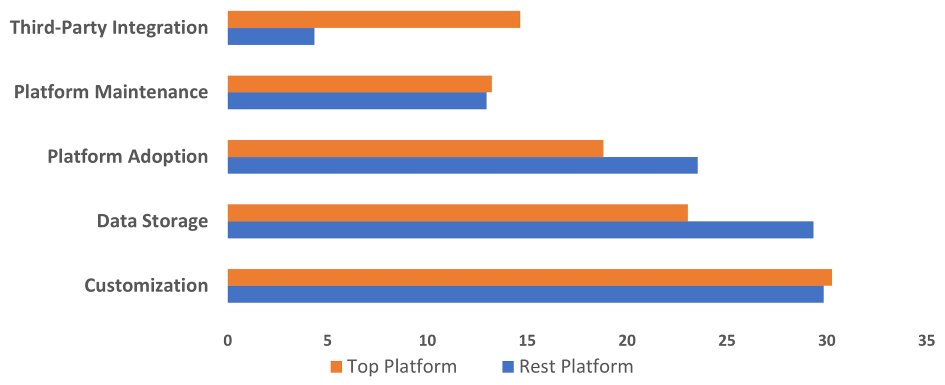


Figure (3.16) The percentage of questions distributed across five topic categories for the top ten LCSD platforms versus the rest of the platforms.

in $Q_{9438695}$. Platform Adoption is a prevalent topic category in the Powerapps, ServiceNow, and Tibco platforms. We also notice that the Data Storage category is quite popular in Filemaker and Lotus Software. Interestingly we see that Zoho Creator LCSD platform around 60% questions belong to Third-party API integration (especially email configuration $Q_{48865565}$). This data sheds light on the popular discussion topics of more recent and earlier LCSD platforms.

3.5.5 The Case of "aggregating" data across multiple LCSD platforms

In this study, we analysed 38 LCSD platforms and these platforms have distinct characteristics and challenges. Our goal is to offer researchers and practitioners a comprehensive overview of the LCSD domain as a whole, as opposed to focusing on a single LCSD platform. Hence, we integrated the data from all of these platforms. For instance, Fig. 3.14

demonstrates that some of the most popular platforms such as Salesforce, Oracle Apex, and Microsoft Powerapps have more questions in SO than other LCSD platforms. Fig. 3.15 demonstrates that questions across these platforms over different topic categories differs slightly. However, Fig. 3.16 shows that questions for Application Customization and Platform Maintenance topic category for top ten platforms vs others remain about the same at around 30% and 13% respectively. Popular platforms have more questions related to Third Party API integration (15%) than others (4%). The top ten platforms have relatively fewer questions (15%) in Data Storage (23% vs 29%) and Platform Adoption (19% vs 24%) Category compared to other platforms. Overall, we found that the observed topics are found across all the platforms that we studied. Given the popularity of some platforms over others, it is understandable that those platforms are discussed are more and as such some platforms can have more coverage (in terms of number of questions) in a topic over other platforms. However, the prevalence of all platforms across each topic shows that the topics are generally well-represented across the platforms.

3.5.6 The application of the taxonomy of challenges

Now we discuss how the findings of this study, i.e., the taxonomy of the discussed challenges can be used to choose a new LCSD platform. The findings of this study can also help practitioners and project managers to get significant insights regarding choosing a particular low-code platform. In Section 3.5.4, we discuss top low-code platforms and we also discuss how the distribution of these topics across different platforms 3.15. From our analysis, new practitioners can have a better understanding of the discussed challenges (3.4.1), their distribution across SDLC phases (3.4.3), and their popularity and difficulty (3.4.4). All these analyses can help low-code practitioners to have a better understanding of the overall domain as well as some potential limitations of supported features across different SDLC phases. For example, Salesforce is one of the most popular LCSD platforms. We can see that Customization (30%), Data Storage (20%), and Platform Adoption (18%) are the three most dominant topic categories. We find that around 40% questions from the Platform Adoption topic categories are asked during Requirement Analysis and Designing SDLC phases. From our topic popularity and difficulty analysis, we find that Platform Adoption and Customization are the most difficult topic categories, and thus we can have a better understanding of the potential challenges of using the Salesforce LCSD platform.

3.5.7 Implications

In Table 3.7, we summarize the core findings of our study and provide recommendations for each findings. The findings from our study can guide the following three stakeholders: (1) LCSD platform Providers to improve the documentation, deployment, and maintenance support, (2) LCSD Practitioners/Developers to gain a better understanding of the trade-offs between rapid development and customization constraints, (3) Community of LCSD Researchers

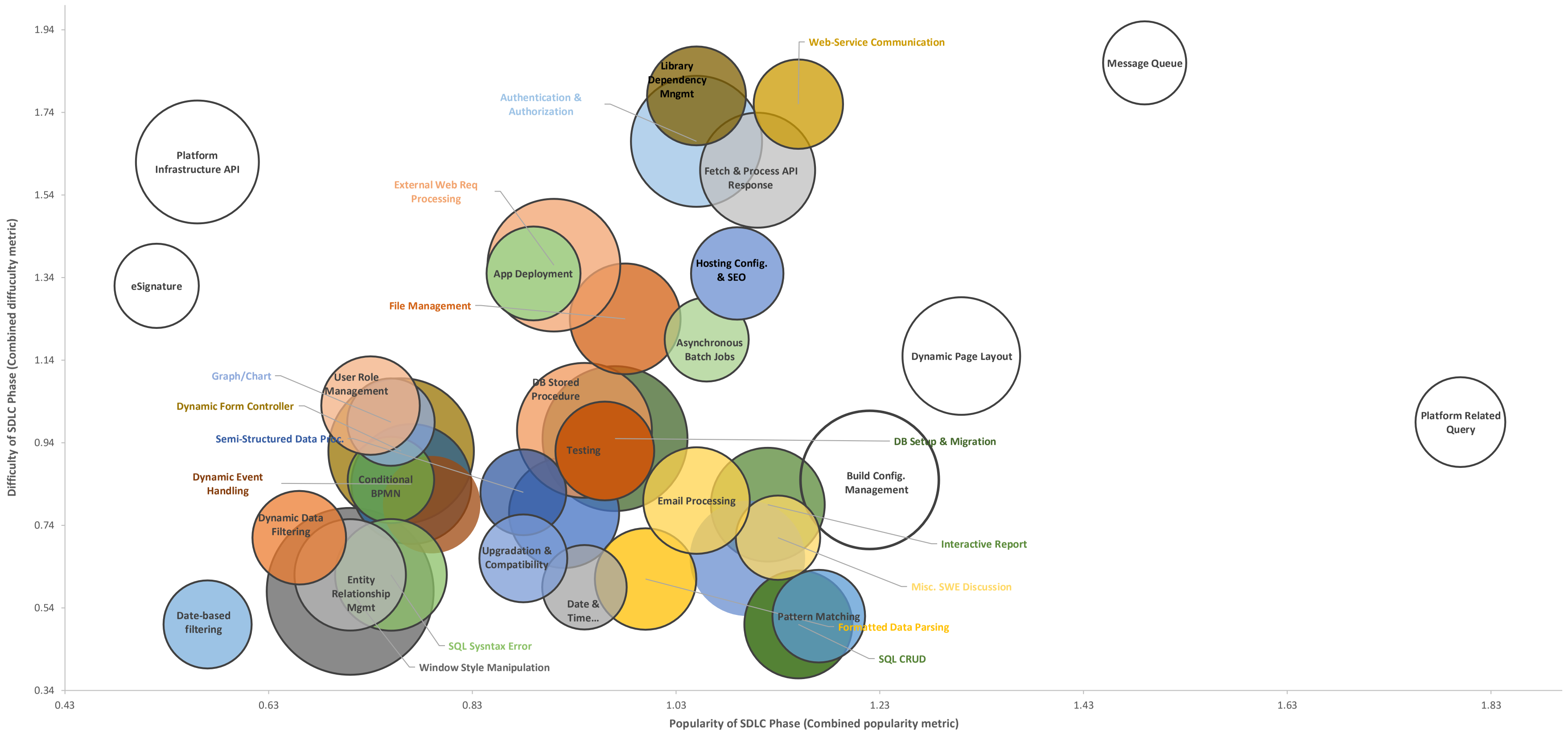


Figure (3.17) The popularity vs. difficulty bubble chart for 40 LCSD-related topics.

Table (3.7) The summary of our core findings and recommendations

RQ	Theme	Core Findings	Recommendations
1	Platform Customization	We identified 40 LCSD topics that fall into five categories. Customization of platform features continues to be the most discussed challenge, referring to the difficulty developers face when seeking to adapt a given feature to a particular situation.	The difficulty with customization can be observed when the LCSD platforms do not offer anyway to change the default interface/functionality. The platforms may offer coding interface where individual components (e.g., a button) can be customized by writing code.
	API Integration	Around 12% of the topics discuss challenges related to the integration of third party apps into an LCSD platform like the integration of REST services.	The LCSD platforms may introduce specialized and programmable interfaces to support the call of a REST service and to be able to process the service output into a format that can be usable within the platform.
2	Platform Adoption	Topics related to customization and platform adoption are being discussed in recent years, because developers are increasingly asking for more documentation and other supporting features as the platforms are being adopted in real-world scenarios.	1 While the official documentation resources can be further improved by the LCSD vendor, automated tools can be investigated to generate documentation by learning of existing adoption of the platforms.
3	How to Use	More than 50% of the questions are How-type, showing the needs for better learning resources.	The official documentation of implementation can be further enhanced by processing the crowd-shared knowledge of the usage discussions of the platforms (e.g., from Stack Overflow)
	Server Setup	What and Why-type questions are most dominant for server setup related.	LCSD platform should provide provide better visualizer and debugger for the practitioners to improve troubleshooting.
4	Development Effort	Implementation is the most dominant SDLC phase (65%). Interestingly we also around one third of the are related to Application Design (17%) and Requirement Analysis & Planning (9%)	LCSD platform providers should take necessary steps to provide better community support in SO to address these challenges. Practitioners should also be aware LCSD platforms can improve the development of traditional software development team but they are not yet panacea for all problems.
5	Deployment vs Maintenance	Questions related to both the deployment-SDLC and Maintenance-SDLC as the most popular and hardest to get accepted answers.	Platform providers should provide better level of abstraction for cloud management, application deployment and monitoring. Educators can provide better resources to learn about cloud platforms.
	Messaging	The topic message queue from platform adoption is the most difficult to get an accepted answer. This topic contains a discussion about the adoption of a general micro-service architecture within LCSD platforms	LCSD platforms can improve the adoption of microservice architecture and the communication between different microservices with better message queuing

& Educators to have a deeper understanding of the significant challenges facing the broader research area to make software development more accessible. We discuss the implications below.

In this empirical study, we infer implications and recommendations based on our observation of practitioners' discussion in SO. So further validation from the developers' survey can provide more insight. However, the diversity of the low-code platforms and topics makes it non-trivial to design a proper survey with representative sample of LCSD practitioners. Therefore, the findings can be used to design multiple LCSD related surveys focusing on different low-code topics and platforms.

LCSD Platform Vendors. In order to better understand the issues of LCSD, we present a bubble chart with difficulty and popularity of different aspects of LCSD such as Topic Category in Figure 4.7, Types of questions in Figure 3.19 and agile SDLC phases in Figure 3.18. These findings coupled with the evolution of LCSD platforms (3.14) and discussions (3.7) shows that Customization and Data Storage related queries are more prevalent, with the majority of these queries occurring during Implementation agile SDLC stage. However, one of our interesting findings is *Platform Adoption* related queries are increasing in popularity. LCSD practitioners find LCSD platform infrastructure and server configuration-related quires tough and popular during the Deployment and Maintenance phase. The top five most challenging topics belong to Platform Adoption and Maintenance topic category.

Many new practitioners make queries regarding LCSD platforms, learning resources, basic application and UI customization, and how to get started with this new emerging technology. Figure 3.17 shows that *Platform Related Query* topic is the most popular among LCSD practitioners. We find that *Documentation* related queries are both top-rated and challenging. Our findings also suggest that many practitioners still face challenges during testing, especially with third-party testing tools like JUnit (in $Q_{9811992}$) and troubleshooting. Consequently, many of the questions on this topic remain unanswered. It reveals that to ensure smooth adoption of the LCSD platforms, providers should provide better and more effective documentation and learning resources to reduce entry-level barriers and smooth out the learning curve.

LCSD Practitioners/Developers. Gartner [107] estimates that by 2022, more than half of the organizations will adapt LCSD to some extent. Additionally, our analysis reveals a rising trend for LCSD approaches, particularly during Covid-19 pandemic (Fig. 3.6). We can also see that new LCSD platforms such as Microsoft Powerapps are gaining many developers' attention. LCSD platform enables practitioners with diverse experience to contribute to the development process even without a software development background. However, our finding shows that practitioners find debugging, application accessibility, and documentation challenges. Hence, the practitioners should take the necessary steps to understand the tradeoffs of LCSD platforms' features deeply. The project manager should adopt specific strategies to customize, debug, and test the application. For example, many practitioners struggle with general Third-Party API integration and database design and query. We find that DevOps-related tasks such as CI/CD, Server configuration, and monitoring-related queries are most challenging to the practitioners. So, a well-functioning

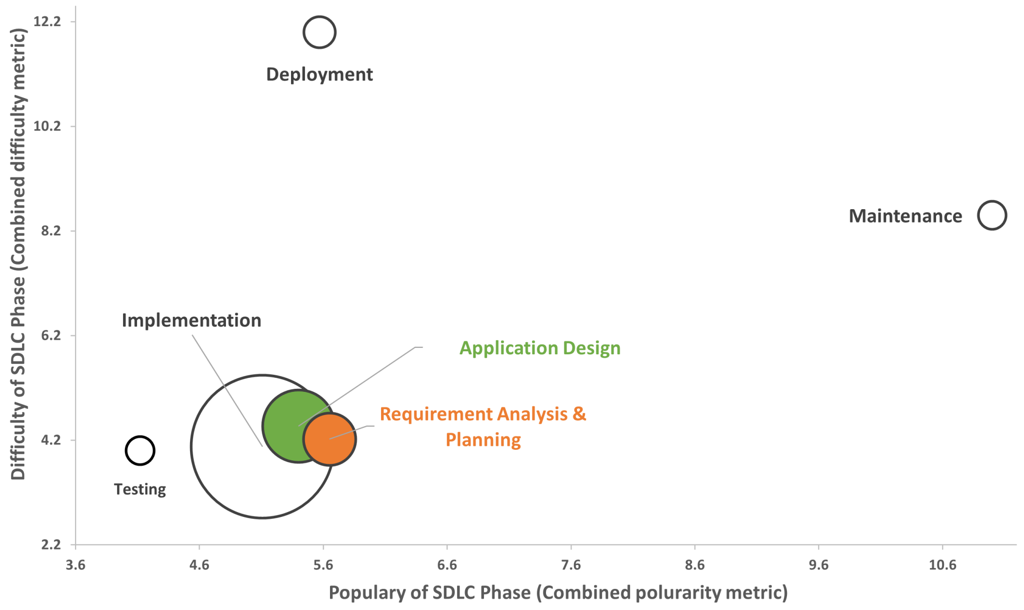


Figure (3.18) The popularity vs. difficulty bubble chart for low-code software development life cycle phases

LCSD team should allocate time and resources to them. It provides valuable insights for project managers to manage resources better (i.e., human resources and development time).

Figure 3.18 shows that Maintenance is the most popular development phase, followed by Deployment, and Testing is the least popular SDLC phase. Similarly, the figure also shows that questions asked in the deployment phase are the most difficult followed by Maintenance. Implementation, Requirement analysis and planning, Application design phase are in the middle range in terms of popularity and difficulty spectrum. Thus, our analysis indicates that LCSD practitioners face more broad and complex application maintenance and deployment-related challenges, on which LCSD platform vendors should concentrate their efforts. This finding can influence the decision-making process of LCSD developers and practitioners like prioritizing their efforts during the design, development, and deployment of software that uses LCSD platforms. For example, if sufficient support or tools are not available for scalable usage and deployment of an LCSD platform, developers may look for alternatives that have better deployment and maintenance support.

As discussed in Section 3.5.6, the findings of this study can also guide project managers to select new LCSD platform. One fundamental shortcoming of LCSD platforms is that their abstraction and feature limitations can make customization and debugging extremely difficult. Additionally, managed cloud platforms make data management and deployability more challenging [165,217]. The findings in this study help to present some strengths and limitations of the overall LCSD paradigm, which complements the findings of other studies [3,13,165,181,217,263]. The analysis could assist LCSD teams in selecting the appropriate LCSD platforms, which is critical for future success.

LCSD Researchers & Educators. The findings of this study have many implications for researchers and ed-

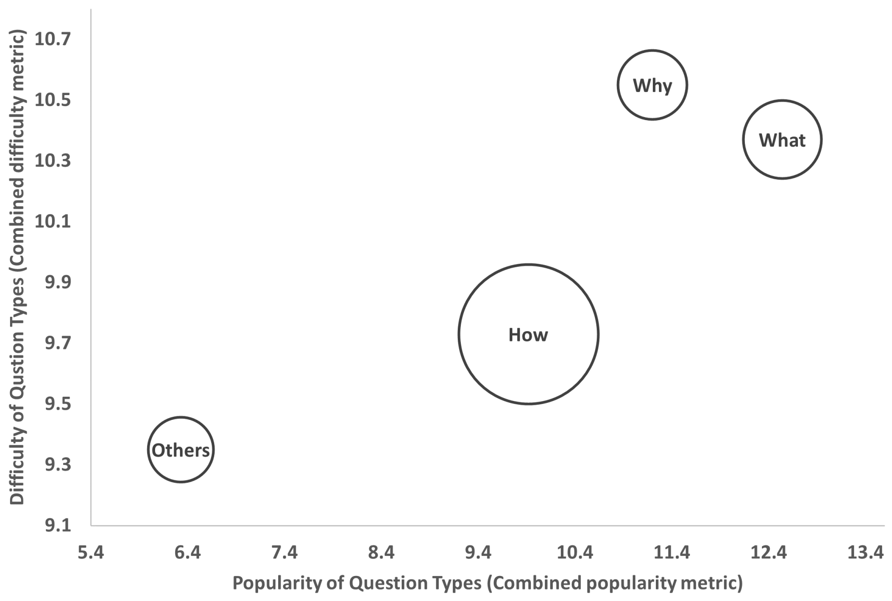


Figure (3.19) The popularity vs. difficulty of different types of LCSD-related questions

ucators of LCSD platforms and the border research community to improve the software development process. We discover that What-type and How-type questions are popular among LCSD practitioners. They also find them challenging because of adequate usable documentation. Thus, practitioners ask questions about certain limits or how to implement certain features, and in the accepted answer, some other user simply points to the official documentation page (e.g., “Domino Data Service API Documentation” in $Q_{59739877}$ and $Q_{5806293}$). Many of the challenges faced by low-code petitioners are similar to traditional software developers. So, researchers from border software engineering domain can contribute to improving aspects such as improving documentation [47,50,142], improving API description usage [251, 252] and make it more accessible to general practitioners. In the Customization and Data Storage topic category, we find practitioners asking help in generic programming queries, database design, and file management. So, research on those topics will also help the adoption of LCSD. Some LCSD platforms provide great in-build support for unit and functional testing. However, we find around *2.1% of questions belong to Testing topic*. Most of these LCSD platforms heavily rely on cloud computing, and thus research improvement of server configuration and library management, i.e., DevOps [281] in general, will aid in better platforms. On the other hand, educators can focus their efforts on making the learning resources on Automatic testing, Server config. and DevOps practices such as CI/CD more accessible to the citizen developers.

Figure 3.19 shows What-type of posts are most popular, followed by Why-type, How-type, and Others-type. Additionally, it demonstrates that the most challenging question type is Why-type, followed by What-type, How-type, and others. So, although How-type questions are most dominant, the details types (i.e., Why-type, What-type) of questions are more popular and challenging. This analysis implies that LCSD practitioners have a harder time finding detailed information regarding different platform features. As a result, LCSD platform providers should improve their documentation. Intuitively, How-type questions can be answered with better documentation for LCSD platforms. Given the official API documentation can often be incomplete and obsolete [143, 253] and given that our research shows that LCSD developers use SO to ask questions about various topics, LCSD researchers can develop techniques and tools to automatically improve the documentation of LCSD platforms by analyzing SO questions and answers. Indeed, existing research efforts show that solutions posted in SO can be used to improve API documentation by supporting diverse development tasks and programming languages [61, 243, 245–247, 249, 250].

We find that the LCSD paradigm's challenges can be different from traditional software development [217]. Simultaneously, researchers can study how to provide better tools for practitioners to customize the application. Security is an open research opportunity for such platforms as a security vulnerability in such platforms or frameworks could compromise millions of applications and users [160]. Researchers can develop better testing approaches to ensure faster development and dependability. Educators can also benefit from the results presented in Table 4.2, 4.2 and Figure 3.17 to prioritize their focus on different topics such as *Library Dependency Mngmt, Web-Service Communication, Asynchronous Batch Jobs, Testing, Dynamic Form Controller*.

3.6 Threats to Validity

Internal validity threats, in our study, relate to the authors' bias while conducting the analysis as we have manually labeled the topics. We mitigate the bias in our manual labeling of topics, types of questions, and LCSD phases by consulting the labels among multiple authors and resolving any conflicts via discussion. Four of the authors actively participated in the labelling process. The first author reviewed the final labels and refined the labels by consulting with the second author.

Construct Validity threats relate to the errors that may occur in data collection, like identifying relevant LCSD tags. To mitigate this, we created our initial list of tags, as stated in Section 4.3, by analyzing the posts in SO related to the leading LCSD platforms. Then we expanded our tag list using state-of-art approach [2, 7, 37, 215]. Another potential threat is the topic modeling technique, where we choose $K = 45$ as the optimal number of topics for our dataset B. This optimal number of topics has a direct impact on the output of LDA. We experimented with different values of K following related works [2, 37]. We used the coherence score and manual examination to find K's optimal value that gives us the most relevant and generalized low-code related topics [2, 10, 254].

External Validity threats relate to the generalizability of our findings. Our study is based on data from developers' discussions on SO. However, there are other forums LCSD developers may use to discuss. We only considered questions and accepted answers in our topic modeling. We also had the option of choosing the best answer. In SO, the accepted answer and best answer may be different. Accepted answer is the one approved by the questioner while the best answer is voted by all the viewers. as discussed in Section 3.5.1 it is quite difficult to detect if an answer is relevant to the question or not. Thus We chose the accepted answer in this study because we believe that the questioner is the best judge of whether the answer solves the problem or not. Even without the unaccepted answers, our dataset contains around 38K posts (27K questions + 11K accepted answers). This also conforms with previous works [2, 10, 254, 275]. Some novice practitioners post duplicate questions, assign incorrect tags, and provide inadequate descriptions, which receive an overall negative score from the community. To ensure that topics contain relevant discussion of high quality, we only use posts with non-negative scores. Nevertheless, we believe using SO's data provides us with generalizability because SO is a widely used Q&A platform for developers. However, we also believe this study can be complemented by including the best answers to the questions in SO, as we discussed earlier, including discussions from other forums, surveying, and interviewing low-code developers.

3.7 Related Work

We previously published a paper at the MSR 2021 based on an empirical study of LCSD topics in SO (see [10]). We compared the findings of this chapter against our previous paper in Section 3.1. Other related work can broadly be divided into two categories: SE (Software Engineering) research on/using (1) low code software development (Section 3.7.1), and (2) topic modeling (Section 4.8.3).

3.7.1 Research on Low Code Software Development and Methodologies

LCSD is a new technology, with only a handful of research papers published in this field. Some research has been conducted on the potential applications of this developing technology in various software applications [105] or for automating business process in manufacturing [263], healthcare [181, 268], Digital transformation [197], Industrial engineering education [3], IoT systems using LCSD [132]. Sipio et al. [77] present the benefits and future potential of LCSD by sharing their experience of building a custom recommendation system in the LCSD platform. Kourouklidis et al. [147] discuss the low-code solution to monitor the machine learning model's performance. Sahay et al. [217] survey LCDP and compare different LCDPs based on their helpful features and functionalities. Khorram et al. [145] analyse commercial LCSD platforms and present a list of features and testing challenges. Zhuang et al. [282] created a low-code platform called EasyFL where researchers and educators can easily build systems for privacy-preserving distributed learning method. Ihirwe et al. [133] analyse 16 LCSD platforms and identifies what IoT application-related features and services each platform provides. All of these studies compare a single LCSD platform and its support and limitations for various sorts of applications [12], rather than taking a holistic picture of the difficulties that the broader community faces.

There are also some studies where researchers proposed different techniques to improve LCSD platform such as Overeem et al. [187] on LCSD platform's impact analysis, Jacinto et al. [135] improve testing for LCSD platforms.

Additionally, there are some studies that describe the difficulties faced by LCSD practitioners. The main research methodology and objectives of these studies, however, are significantly different from this study. Yajing et al. [165] analyse the LCSD platform's characteristics including programming languages used, major implementation units, supporting technologies, applications being developed, domains, etc., along with the benefits, limitations, and challenges by collecting relevant posts from SO and Reddit. In this study, we use tag-based approach to find relevant LCSD-related posts which is much more reliable than text-based searching. Furthermore, the SO related discussion used in this study is significantly larger and our research objective about LCSD platforms challenges are quite different. Timothy et al. [152] discuss experiences with several low-code platforms and provide recommendations focusing on low-code platforms enabling scaling, understandability, documentability, testability, vendor-independence, and the overall user experience for developers as end-users who do some development. Danial et al. [71] and ALSAADI et al. [13] Surveyed on factors hindering the widespread adaptation of LCSD by interviewing LCSD developers or conducting a survey. To the best of our knowledge, ours is the first empirical study of LCSD platforms based on developers' discussions from Stack Overflow, and hence our findings complement those of other studies.

3.7.2 Topic Modeling in Software Engineering

Our motivation to use topic modeling to understand LCSD discussions stems from existing research in software engineering that shows that topics generated from textual contents can be a good approximation of the underlying *themes* [64, 231, 232]. Topic models are used recently to understand software logging [154] and previously for diverse other tasks, such as concept and feature location [67, 202], traceability linking (e.g., bug) [24, 206], to understand software and source code history evolution [129, 235, 236], to facilitate code search by categorizing software [237], to refactor software code base [46], as well as to explain software defect [63], and various software maintenance tasks [230, 231]. The SO posts are subject to several studies on various aspects of software development using topic modeling, such as what developers are discussing in general [43] or about a particular aspect, e.g., concurrency [7], big data [37], chatbot [2].

Table (3.8) Comparing the popularity and difficulty metrics of different domains

Type	Metrics	LCSD	IoT	Big Data	Chatbot	Security	Mobile	Concurrency
P	# Posts	33,766	53,173	125,671	3,890	94,541	1,604,483	245,541
	Avg View	1330.6	1,320.3	1,560.4	512.4	2,461.1	2,300.0	1,641
	Avg Favorite	1.2	1.5	1.9	1.6	3.8	2.8	0.8
	Avg Score	0.7	0.8	1.4	0.7	2.7	2.1	2.5
D	% W/o Acc. Ans	41%	64%	60.3%	67.7%	48.2%	52%	43.8%
	Med Hrs to Acc.	5.7	2.9	3.3	14.8	0.9	0.7	0.7

Table (3.9) Comparative analysis of question types across different domains

Question Type	**How**	**What**	**Why**	**Others**
LCSD	55.6%	17.9%	14%	12.5%
IoT	47.3%	37.9%	20%	8.3%
Chatbot	61.8%	11.7%	25.4%	1.2%

In particular, SO posts have been used in various studies where the researchers analysed topics for that particular domains. For instance, SO posts has been used to study developers challenges in IoT [254], big data [37], chatbots [2] and so on. The distributions and the nature of these posts differs. As SO is arguably the most popular public forum for developers, the analysis of these domains' characteristics may help us identify the SO community better. Therefore, a systematic analysis of these domains is interesting. Following related studies [254], we use six metrics in this study: (1) Total #posts, (2) Avg views, (3) Avg favorite, (4) Avg score, (5) Percentage of questions without accepted answers, (6) Median hours to get accepted answers per domain. The first four metrics are popularity metrics and the last two are difficulty metrics.

In this study, we do not replicate the findings of the original study in our dataset. Rather we only report the findings from the original study. So following related work [254], we compared our LCSD-related discussions with other five domains: IoT [254],big data [37], security [277], mobile apps [214], chatbots [2] and concurrency [7].

Table 3.8 provides an overview of the seven metrics. We can see that it has a greater number of SO posts than chatbot domains but fewer than the other five domains. There are two other studies on Blockchain [260] and deep learning [124] where the total number of posts are 32,375 and 25,887, respectively. However, these two studies did not report the other metrics, so they are excluded from the Table. Although the LCSD-related discussion may have fewer posts than these other domains, as discussed in RQ3, this number is increasing rapidly.

We can also observe that the LCSD domain shows similarities with IoT, Concurrency domain in terms of Avg. View count. Security and Mobile domain seem most popular in terms of Avg. Favourite count and LCSD rank lower in this metric. LCSD domains most resemble with IoT domain in terms of Avg. View, Avg. Favourite and Avg. Score. In terms of difficulty metrics percentage of posts without accepted answers, LCSD domain ranks lowest, which is good. However, it takes much longer to get accepted answers(0.7 hours for Mobile, 2.9 for IoT). Only the chatbot domain requires more time to get accepted answers (i.e., 5.3 hours in LCSD vs 14.8 hours in chatbot).

We further discuss the LCSD, IoT, and Chatbot domains with the distribution of different types of questions in Table 3.9. We find that LCSD domain is in the middle of IoT and Chatbot domains in terms of How-type (57%) compared to IoT (47%) and chatbot (62%). This signifies that LCSD domain practitioners ask more about implementation-related questions. In the Why-type question percentage of LCSD-related domains is lowest (14%) compared with IoT (20%) and chatbot (25%). This suggests that practitioners in the LCSD domain enquire about relatively modest troubleshooting issues. For What-type, we notice that the IoT domain dominates with 38% of questions, compared to the LCSD domain's 12%. Practitioners of LCSD are less inquisitive about domain architecture and technologies compared to IoT domain. As a result of these analyses, we can see that LCSD domain practitioners exhibit several traits that distinguish them from practitioners in other domains, which LCSD vendors and educators should take into account.

3.8 Summary

Low Code Software Development (LCSD) is a novel paradigm for developing software applications utilizing visual programming with minimum hand-coding. We present an empirical study that provides insights into the types of discussions low-code developers discuss in Stack Overflow (SO). We find 40 low-code topics in our dataset of 33.7K SO posts (question + accepted answers). We collected these posts based on 64 SO tags belonging to the popular 38 LCSD platforms. We categorize them into five high-level groups, namely Application Customization (30% Questions, 11 Topics), Data Storage (25% Questions, 9 Topics), Platform Adoption (20% Questions, 9 Topics), Platform Maintenance (14% Questions, 6 Topics), and Third-Party Integration (12% Questions, 5 Topics). The Platform Adoption topic category has recently gained popularity. Platform Related Query and Message Queue topics from this category are the most popular. On the other hand, We also find that practitioners find Platform Adoption and Maintenance related topics most challenging. How-type questions are the most common, but our research reveals that practitioners find what-type and why-type questions more difficult. Despite extensive testing, deployment, and maintenance support, our analysis shows that server configuration, data migration, and system module upgrading-related queries are widespread and complex to LCSD practitioners. Despite significant testing, deployment, and maintenance support, our analysis finds numerous complex queries regarding server configuration, data migration, and system module updating. Our analysis finds that better tutorial-based documentation can help solve many of these problems. We also find that during the Covid-19 pandemic, LCSD platforms were very popular with developers, especially regarding dynamic form-based applications. We hope that these findings will help various LCSD stakeholders (e.g., LCSD platform vendors, practitioners, and SE researchers) take necessary actions to address the various LCSD challenges. Since the growth indicates that this technology is likely to be widely adopted by various companies for their internal and customer-facing applications, platform providers should address the prevailing developers' challenges. Our future work will focus on (1) getting developers' feedback on our study findings based on surveys and developer interviews, and (2) developing tools and techniques to address the challenges in the LCSD platforms we observed automatically.

Chapter 4

Challenges and Barriers of Using Low Code Software for Machine Learning Application Development

From the background study of Chapter 2, we see the overall low-code software development is divided into two parts: traditional software application and machine learning application. In the previous Chapter 3, we reported the challenges of traditional LCSD practitioners challenges. This chapter aims to conduct an empirical study of the challenges of low-code ML, i.e., AutoML practitioners. This research presents an empirical study of around 14k posts (questions + accepted answers) from Stack Overflow (SO) that contained AutoML-related discussions. Software developers frequently use the online developer Q&A site SO to seek technical assistance. We observe a growing number of AutoML-related discussions in SO. We use LDA Topic Modeling to determine the topics discussed in those posts. Additionally, we examine how these topics are spread across the various Machine Learning Life Cycle (MLLC) phases and their popularity and difficulty. This study offers several interesting findings. First, we find 13 AutoML topics that we group into four categories. The MLOps topic category (43% questions) is the largest, followed by Model (28% questions), Data (27% questions), and Documentation (2% questions). Second, Most questions are asked during Model training (29%) (i.e., implementation phase) and Data preparation (25%) MLLC phase. Third, AutoML practitioners find the MLOps topic category most challenging, especially topics related to model deployment & monitoring and Automated ML pipeline. Fourth, the MLOps topic category and Model deployment and Monitoring phase are more predominant and popular in cloud-based AutoML solution and Model topic categories. In contrast, the Model evaluation phase is more dominant and popular in non-cloud AutoML solutions. These findings affect all three AutoML stakeholders: AutoML researchers, AutoML service vendors, and AutoML developers. Academia and Industry collaboration can improve different aspects of AutoML, such as better DevOps/deployment support and tutorial-based documentation.

4.1 Introduction

With the rise of automation and digitization, the term "big data" has become ubiquitous across a wide range of industries. The rapid growth of cloud computing and tremendous progress in machine learning has attracted businesses to hire ML engineers to make the most value out of their data. ML has grown in popularity as a solution to various software engineering issues, including speech recognition, image processing, natural language processing etc. Nevertheless, The development of an ML model requires significant human expertise. It is necessary to use intuition based on prior knowledge and experience to determine the best ML algorithm/architecture for each dataset. This high demand of ML expertise coupled with the shortage of ML developers and the repetitive nature of many ML pipeline processes have inspired the adoption of low-code machine learning (i.e., AutoML) tools/platforms.

The primary goal of low-code AutoML services is to reduce the manual efforts of developing different ML pipelines, which thus helps in accelerating their development and deployment. This is necessary due to the fact that a business may have a specific business requirement and the domain experts/stakeholders in the business know it, but they lack the skills to develop an ML application. Indeed, for ML development, there are often primarily two key users as follows.

1. **Domain Experts** are knowledgeable about the problem domain (e.g., rain forecasting, cancer diagnosis, etc.) where the ML is applied. So that they have a deep understanding of the scope of the problem and the dataset.
2. **ML Experts** are experienced with the nuances of the ML model. They have a greater understanding and experience in selecting the appropriate ML algorithm/architecture, engineering features, training the model, and evaluating its performance.

Both of the domain and ML experts are equally important for the success of the ML-based solution development, and are often complementary to each other during the design and development of the solutions. AutoML aims to make ML more accessible to domain experts by providing end-to-end abstraction from data filtering to model designing & training to model deployment & monitoring. AutoML is becoming an essential topic for software engineering researchers as more and more AutoML applications are developed and deployed. AutoML research is progressing rapidly as academia and industry work collaboratively by making ML research their priority. For example, currently, in at least some experimental settings, AutoML tools yield better results than manually designed models by ML experts [167] (c.f. background of AutoML in Section 5.3).

Machine learning practitioners face many challenges because of the interdisciplinary nature of the ML domain [14]. They require not only the software engineering expertise to configure and set up new libraries but also in data visualization, linear algebra and statistics. There have been quite some studies [134, 219] on the challenges of deep learning and ML tools, but there are no formal studies on the challenges AutoML practitioners have asked about on public forums. There are also studies on developers' discussion on machine learning domain [14, 42, 131] and deep learning frameworks [65,70,124,134,280], but *there are no studies on low-code, i.e., AutoML tools/platforms (c.f. related work in Section 4.8).*

The online developer forum Stack Overflow (SO) is the most popular Q&A site with around 120 million posts and 12 million registered users [188]. Several research has conducted by analyzing developers' discussion of SO posts (e.g., traditional low-code practitioners discussion [10, 11], IoT [254], big data [37], blockchain [260], docker developers challenges [125], concurrency [7], microservices [41]) etc. *The increasing popularity and distinctive character of low-code ML software development approaches (i.e., AutoML tools/frameworks) make it imperative for the SE research community to explore and analyze what practitioners shared publicly.*

To that end, in this chapter, we present an empirical study of 14.3K SO posts (e.g., 10.5k Q + 3.3k Acc Ans) relating to AutoML-related discussion in SO to ascertain the interest and challenges of AutoML practitioners (c.f. study methodology in Section 4.3). We use SOTorrent dataset [97] and Latent Dirichlet Allocation (LDA) to systematically analyze practitioners' discussed topics similar to other studies [2,7,10,11,254]. We explore the following four research questions by analyzing the dataset (c.f. results in Section 4.4).

RQ1. What topics do practitioners discuss regarding AutoML services on SO? As low-code machine learning (i.e., AutoML) is an emerging new paradigm, it is vital to study publicly shared queries by AutoML practitioners on a Q&A platform such as SO. We extract 14.3K AutoML related SO posts and apply the LDA topic modelling method [52] to our dataset. We find a total of 13 AutoML-related topics grouped into four categories: MLOps (43% Questions, 5 topics), Model (28% Questions, 4 topics), Data (27% Questions, 3 topics), and Documentation (2% Questions, topic). We find that around 40% of the questions are related to supported features of a specific AutoML solution providers, around 15% of questions are related to model design and development, and 20% of questions are related to data pre-processing and management. We find relatively fewer questions on programming, and details configuration on ML algorithm as AutoML aims to provide a higher level abstraction over data processing and model design and development pipeline. However, AutoML practitioners still struggle with development system configuration and model deployment.

RQ2. How AutoML topics are distributed across machine learning life cycle phases? AutoML aims to provide an end-to-end pipeline service to streamline ML development. The successful adoption of this technology largely depends on how effective this is on different machine learning life cycle (MLLC) phases. So, following related studies [9, 14, 134], we manually analyze and annotate 348 AutoML questions into one of six ML life cycle stages by taking statically significant questions from all the four topic categories. We find that Model training, i.e., the implementation phase is the most dominant (28.7% questions), followed by Data Preparation (24.7% questions) and Model Designing (18.7% questions).

RQ3. What AutoML topics are most popular and difficult on SO? From our previous research questions, we find that AutoML practitioners discuss diverse topics in different stages of machine learning development. However, some of these questions are popular and have larger community support. We find that AutoML practitioners find the MLOps topic category most popular and challenging. AutoML practitioners find the Requirement Analysis phase the most challenging and popular, followed by the model deployment & monitoring MLLC phase. We find that AutoML practitioners find Model Deployment & Load topic to be most challenging regarding the percentage of questions without accepted answers and median hours required to get accepted answers. We also find that questions related to improving the performance of AutoML models are most popular among AutoML practitioners regarding average view count and average score.

RQ4. How does the topic distribution between cloud and non-cloud AutoML services differ? Our analyzed dataset contains both cloud-based and non-cloud-based AutoML services. Cloud-based solutions provides an end-to-end pipeline for data collecting to model operationalization, whereas non-cloud-based services offer greater customizability for data preparation and model development. We find that, in our dataset around 65% SO questions belong to cloud and 35% belong to non-cloud based AutoML solution. Cloud-based AutoML solution predominate over Model Deployment and Monitoring phase (82%) and non-cloud based AutoML solution predominate over Model Evaluation phase. MLOps topic category (i.e., 80%) is predominated over cloud-based AutoML solutions, while Model (i.e., 59%) topic category is predominated by non-cloud based AutoML solutions.

Our study findings offer valuable insights to AutoML researchers, service providers, and educators regarding what aspect of AutoML requires improvement from the practitioners' perspective (c.f. discussions and implications in Section 5.5). Specifically, our findings can enhance our understanding of the AutoML practitioners' struggle and help the researchers and platform vendors better focus on the specific challenges. For example, AutoML solutions lack better support for deployment, and practitioners can prepare for potentially challenging areas. In addition, all stakeholders and practitioners of AutoML can collaborate to provide enhanced documentation and tutorials. The AutoML service vendors can better support model deployment, monitoring and fine-tuning.

Replication Package: The code and data are shared in `https://github.com/disa-lab/Thesis-Alamin-MSc/tree/main/Chapter_4`

4.2 Background

4.2.1 AutoML as Low-code Tool/Platform for ML

Recent advancements in machine learning (ML) have yielded highly promising results for a variety of tasks, including regression, classification, clustering, etc., on diverse dataset types (e.g., texts, images, structured/unstructured data).

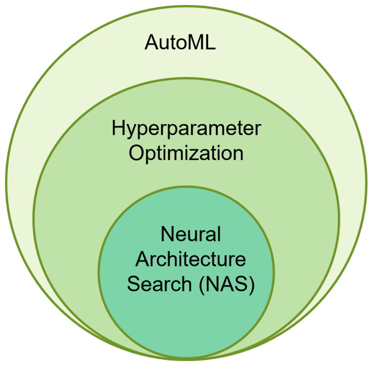


Figure (4.1) An overview of AutoML for discovering the most effective model through Neural Architecture Search (NAS) [31]

The development of an ML model requires significant human expertise. Finding the optimal ML algorithm/architecture for each dataset necessitates intuition based on past experience. ML-expert and domain-expert collaboration is required for these laborious and arduous tasks. The shortage of ML engineers and the tedious nature of experimentation with different configuration values sparked the idea of a low-code approach for ML. This low-code machine learning solution seeks to solve this issue by automating some ML pipeline processes and offering a higher level of abstraction over the complexities of ML hyperparameter tuning, allowing domain experts to design ML applications without extensive ML expertise. It significantly increases productivity for machine learning practitioners, researchers, and data scientists. The primary goal of low-code AutoML tools is to reduce the manual efforts of different ML pipelines, thus accelerating their development and deployment.

In general terms, a machine learning program is a program that can learn from experience, i.e., data. In the traditional approach, a human expert analyses data and explores the search space to find the best model. AutoML [73, 126] aims to democratize machine learning to domain experts by automating and abstracting machine learning-related complexities. It aims to solve the challenge of automating the Combined Algorithm Selection and Hyper-parameter tuning (CASH) problem. AutoML is a combination of automation and ML. It automates various tasks on the ML pipeline such as data prepossessing, model selection, hyper-parameter tuning, and model parameter optimization. They employ various types of techniques such as grid search, genetics, and Bayesian algorithms. Some AutoML services also help with data visualization, model interpretability, and deployment. It helps non-ML experts develop ML applications and provides opportunities for ML experts to engage in other tasks [109]. The lack of ML experts and exponential growth in computational power make AutoML a hot topic for academia and the industry. Research on AutoML research is progressing rapidly; in some cases, at least in the experimental settings, AutoML tools are producing the best hand-designed models by ML experts [167].

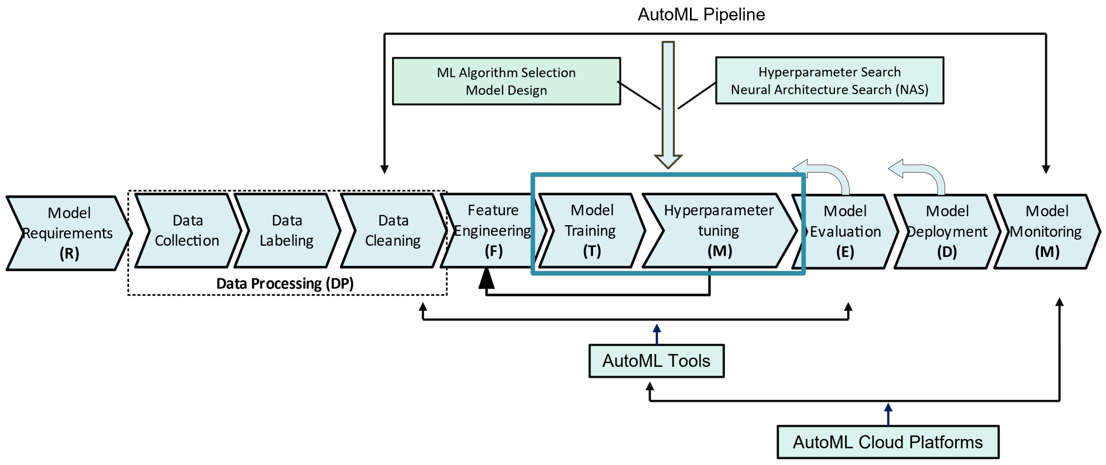


Figure (4.2) An overview of traditional ML pipeline vs AutoML pipeline.

4.2.2 AutoML Approaches

AutoML approach can be classified into two categories

1. AutoML for traditional machine learning algorithms. It focuses on data pre-processing, feature engineering (i.e., finding the best set of variables and data encoding technique for the input dataset), ML algorithm section, and hyperparameter tuning.

2. AutoML for deep learning algorithms: this includes Neural architecture search (NAS), which generates and assesses a large number of neural architectures to find the most fitting one by leveraging reinforcement learning [233] and genetic algorithm [176].

Figure 4.1 provides a high-level overview of AutoML for NAS and hyper-parameter optimization. The innermost circle represents the NAS exploration for the DL models, and the middle circle represents the hyper-parameter optimization search for both NAS and traditional ML applications.

4.2.3 AutoML Services

Depending on their mode of access/delivery, currently availalbe AutoML tools can be used over the cloud or via a stand-alone application in our desktop or internal server computers.

AutoML Cloud Service/Platforms. Large cloud providers and tech businesses started offering Machine Learning as a Service projects to make ML even more accessible to practitioners due to the rising popularity of AutoML tools. Some of these platforms specialize in various facets of AutoML, such as structured/unstructured data analysis, computer vision, natural language processing, and time series forecasting. In 2016, Microsoft released AzureML [234] runs on top of Azure cloud and assists ML researchers/engineers with data processing and model development. H2O Automl [150] was released in 2016, followed by H2O-DriverlessAI [120] in 2017. It is a customizable data science platform with automatic feature engineering, model validation and selection, model deployment, and interpretability. In 2017 Google released Google Cloud AutoML [115] that provides end-to-end support to train the custom model on the custom dataset with minimal effort. Some other notable cloud platforms are Darwin (2018) [72] AutoML cloud platform for data science and business analytics, TransmogrifAI (2018) [240] runs on top of Salesforce's Apache Spark ML for structured data. This cloud-based AutoML platform enables end-to-end data analytics and AI solutions for nearly any sector.

AutoML Non-cloud Service (Tools/Library). The initial AutoML tools were developed in partnership with academic researchers and later by startups and large technology corporations [242]. Researchers from the University of British Columbia and Freiburg Auto-Weka (2013) [146], which is one of the first AutoML tools. Later, researchers from the University of Pennsylvania developed TPOT (2014) [184], and researchers from the University of Freiburg released Auto-Sklearn (2014) [100]. These three AutoML tools provide a higher level abstraction over the popular ML library "SciKit-Learn" (2007) [195]. A similar research effort was followed to provide an automated ML pipeline over other popular ML libraries. University of Texas A&M University developed Auto-Keras (2017) [136] that provides runs on top of Keras [117] and TensorFlow [223]. Some of the other notable AutoML tools are MLJar (2018) [200], DataRobot (2015) [74], tool named "auto_ml" (2016) [28]. These AutoML tools provide a higher level of abstraction over traditional ML libraries such as TensorFlow, Keras, and Scikit-learn and essentially automate some ML pipeline steps (i.e., algorithm selection and hyper-parameter turning).

In Figure 4.2, we summarize the ML pipeline services offered by AutoML services. Traditional ML pipeline consists of various steps such as Model requirement, Data processing, feature engineering, model deigning, model evaluation, deployment, and monitoring [9, 15]. AutoML solutions aim to automate various stages of these pipelines, from data cleaning to Model deployment [242] (Fig. 4.2). AutoML non-cloud solutions (i.e., tools/frameworks) mainly focus on automating different data filtering, model selection, and hyperparameter optimization. AutoML cloud platforms encapsulate the services of AutoML tools and provide model deployment and monitoring support. They usually provide the necessary tools for data exploration and visualization.

4.3 Study Data Collection and Topic Modeling

In this Section, we discuss our data collection process to find AutoML-related discussion, i.e., posts (Section 4.3.1). Then, we discuss in detail our data pre-processing and topic modeling steps on these posts (Section 4.3.2).

4.3.1 Data Collection

We collect AutoML related SO posts in the following three steps: (1) Download SO data dump, (2) Identify AutoML-related tag list, and (3) Extract AutoML-related posts using our AutoML tag list. We describe the steps in detail below.

Step 1: Download SO data dump. For this study, we use the most popular Q&A site, Stack Overflow (SO), where practitioners from diverse backgrounds discuss various software and programming-related issues [188]. First, we download the latest SO data dump [97] of June 2022, available during the start of this study. Following related studies [2, 10, 254], we use the contents of the "Post.xml" file, which contains information about each post like the post's unique ID, title, body, associated tags, type (Question or Answer), creation date, favorite count, view-count, etc. Our data dump includes developers' discussions of 14 years from July 2008 to June 2022 and contains around 56,264,787 posts. Out of them, 22,634,238 (i.e., 40.2%) are questions, 33,630,549 (i.e., 59.7%) are answers, and 11,587,787 questions (i.e., 51.19%) had accepted answers. Around 12 million users from all over the world participated in the discussions.

Each SO post contains 19 attributes, and some of the relevant attributes for this study are: (1) Post's unique Id, and creation time, (2) Post's body with problem description and code snippets, (3) Post's score, view, and favorite count, (4) Tags associated with a post, (5) Accepted answer Id.

Step 2: Identify AutoML tags. We need to identify AutoML related SO tags to extract AutoML-related posts, i.e., practitioners' discussions. We followed a similar procedure used in prior work [2, 7, 10, 161, 254, 260] to find relevant SO tags. In Step 1, we identify the initial AutoML-related tags and call them T_{init}. In Step 2, we finalize our AutoML tag list following approaches of related work [37, 277]. Our final tag list T_{final} contains 41 tags from the top 18 AutoML service providers. We discuss each step in detail below.

(1) Identifying Initial AutoML tags. Following our related work [10], first, we compile a list of top AutoML services. First, we make a query in google with the following two search terms "top AutoML tools" and "top AutoML platforms". We select the first five search results for each query that contain various websites ranked the best AutoML tools/platforms. The full list of these websites is available in our replication package. So, from these ten websites and the popular technological research website Gartner[1] we create a list of 38 top AutoML solutions. Then for each of these AutoML platforms, we search for SO tags. For example, We search for "AutoKeras" via the SO search engine. We find a list of SO posts discussing the AutoKeras tool. We compile a list of potential tags for this platform. For example, we notice most of these questions contain "keras" and "auto-keras" tags. Then, we manually examine the metadata of these tags [2]. For example, the metadata for the "auto-keras" tag says, "Auto-Keras is an open source software library for automated machine learning (AutoML), written in python. A question tagged auto-keras should be related to the Auto-Keras Python package." The metadata for the "keras" tag says, "Keras is a neural network library providing a high-level API in Python and R. Use this tag for questions relating to how to use this API. Please also include the tag for the language/backend ([python], [r], [tensorflow], [theano], [cntk]) that you are using. If

[1] https://www.gartner.com/reviews/market/data-science-machine-learning-platforms
[2] https://meta.stackexchange.com/tags

you are using tensorflow's built-in keras, use the [tf.keras] tag.". Therefore, we choose the "auto-keras" tag for the "AutoKeras" AutoML library. Not all AutoML platforms have associated SO tags; thus, they were excluded. For example, for AutoFolio [27] AutoML library, there are no SO tags; thus, we exclude this from our list. This way, we find 18 SO tags for 18 AutoML services and call it T_{init}. The final AutoML solutions and our initial tag list are available in our replication package.

(2) Finalizing AutoML-related tags. Intuitively, there might be variations to tags of 18 AutoML platforms other than those in T_{init}. We use a heuristic technique from related previous works [10, 37, 277] to find the other relevant AutoML tags. First, we denote the entire SO data as Q_{all}. Second, we extract all questions Q that contain any tag from T_{init}. Third, we create a candidate tag list $T_{candidate}$ using the relevant tags in the questions Q. Fourth, we analyze and select significantly relevant tags from $T_{candidate}$ for our AutoML discussions. Following related works [10, 11, 37, 254, 277], we compute relevance and significance for each tag t in $T_{candidate}$ with respect to Q (i.e., the extracted questions that have at least one tag in T_{init}) and Q_{all} (i.e., our data dump) as follows,

$$(Significance)\ S_{tag} = \frac{\#\ of\ ques.\ with\ the\ tag\ t\ in\ Q}{\#\ of\ ques.\ with\ the\ tag\ t\ in\ Q_{all}}$$

$$(Relevance)\ R_{tag} = \frac{\#\ of\ questions\ with\ tag\ t\ in\ Q}{\#\ of\ questions\ in\ Q}$$

A tag t is significantly relevant to AutoML if the S_{tag} and R_{tag} are higher than a threshold value. Similar to related study [11, 254], we experimented with a wide range of values of $S_{tag} = \{0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35\}$ and $R_{tag} = \{0.001, 0.005, 0.010, 0.015, 0.020, 0.025, 0.03\}$. From our analysis, we find that we increase S_{tag} and R_{tag} the total number of recommend tags decreases. For example, we find that for $S_{tag} = .05$ and $R_{tag} = 0.001$ the total number of recommended tags for AutoML is 30 which is highest. However, not all these recommended tags are AutoML-related. For example, "amazon-connect" tag's $S_{tag} = 0.24$ and $R_{tag} = 0.006$ and this tag is quite often associated with questions related to other AutoML tags such as "aws-chatbot", "amazon-machine-learning" but it mainly contains discussion related to AWS cloud based contact center solutions rather than AutoML related discussion. So, we remove this from our final tag list. Similarly, we find some other tags such as "splunk-formula", "amazon-ground-truth" etc are frequently correlated with other AutoML platform tags, although they do not contain AutoML related discussions. After manually analysing these 31 tags we find that 23 new tags are relevant to AutoML-related discussions. So, after combining with out initial taglist, i.e., T_{init}, our final tag list T_{final} contains 41 significantly relevant AutoML-related tags which are:

Final Tag List T_{final} = { 'amazon-machine-learning', 'automl', 'aws-chatbot', 'aws-lex', 'azure-machine-learning-studio', 'azure-machine-learning-workbench', 'azureml', 'azureml-python-sdk', 'azuremlsdk', 'driverless-ai', ‘ensemble-learning’, ‘gbm’, ‘google-cloud-automl-nl’, ‘google-cloud-vertex-ai’, ‘google-natural-language’, ‘h2o.ai’, ‘h2o4gpu’, ‘mlops’, ‘sparkling-water’, ‘splunk-calculation’, ‘splunk-dashboard’, ‘splunk-query’, ‘splunk-sdk’, ‘amazon-sagemaker’, ‘tpot’, ‘auto-sklearn’, ‘rapidminer’, ‘pycaret’, ‘amazon-lex’, ‘auto-keras’, ‘bigml’, ‘dataiku’, ‘datarobot’, ‘google-cloud-automl’, ‘h2o’, ‘mljar’, ‘splunk’, ‘transmogrifai’, ‘ludwig’, ‘azure-machine-learning-service’, ‘pycaret’}

Step 3: Extracting AutoML related posts. Hence, our final dataset B contained 14,341 posts containing 73.7% Questions (i.e., 10,549 Q) and 26.3% Accepted Answers (i.e., 3,792).

4.3.2 Topic Modeling

We produce AutoML topics from the extracted posts in three steps: (1) Preprocess the posts, (2) Find the optimal number of topics, and (3) Generate topics. We discuss the steps in detail below.

Step 1. Preprocess the posts. For each post text, we remove noise using the technique in related works [2,10,11,37,43, 254]. First, we remove the code snippets from the post body, which is inside <code></code> tag, HTML tags such as (<p></p>, <a></a>, <li></li> etc.), and URLs. Then we remove the stop words such as “am”, “is”, “are”, “the”, punctuation marks, numbers, and non-alphabetical characters using the stop word list from MALLET [168], NLTK [162]. After this, we use porter stemmer [205] to get the stemmed representations of the words, e.g., “waiting”, “waits” - all of which are stemmed to the base form of the word “wait”.

Step 2. Finding the optimal number of topics. After the prepossessing, we use Latent Dirichlet Allocation [52] and the MALLET tool [168] to find out the AutoML-related topics in our SO discussions. We follow similar studies using topic modeling [2, 10, 11, 22, 24, 37, 254, 277] in SO dataset. Our goal is to find the optimal number of topics K for our AutoML dataset B so that the *coherence* score is high, i.e., encapsulation of underlying topics is more coherent. We use Gensim package [208] to determine the coherence score following previous research [10,213,244]. We experiment with different values of K that range from {5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70} and for each value, we run MALLET LDA on our dataset for 1000 iterations [11, 37, 254]. Then we observe how the coherence score changes with respect to K. As LDA topic modeling has some inherited randomness, we ran our experiment 3 times and found that we got the highest coherence score for $k = 15$. Choosing the right value of K is crucial because multiple real-world topics merge for smaller values of K, and for a large value of K, a topic breaks down. MALLET uses two hyper-parameters, α, and β, to distribute words and posts across the generated topics. Following the previous works [7, 10, 37, 39, 215, 254], in this study we use the standard values $50/K$ and 0.01 for hyper-parameters α and β in our experiment.

Step 3. Generating topics. Topic modeling is a systematic approach to extracting a set of topics by analyzing a collection of documents without any predefined taxonomy. Each document, i.e., the post, has a probability distribution of topics, and every topic has a probability distribution of a set of related words [43]. We generate 15 topics using the above LDA configuration on our AutoML dataset *B*. Each topic model provides a list of the top *N* words and a list of *M* posts associated with the topic. A topic in our context comprises the 30 most commonly co-related terms, which indicate an AutoML development-related concept. Each post has a correlation score between 0 to 1, and following the previous work [10, 11, 254, 260], we assign a document, i.e., a post with a topic that corresponds most.

4.4 Empirical Study

We answer the following four research questions by analyzing the topics we found in our 14.3K AutoML related posts in Stack Overflow (SO).

RQ1. What topics do practitioners discuss regarding AutoML services on SO?

RQ2. How are the AutoML topics distributed across machine learning life cycle phases?

RQ3. What AutoML topics are most popular and difficult on SO?

RQ4. How does the distribution of the topics differ between cloud and non-cloud AutoML services?

4.4.1 RQ1. What topics do practitioners discuss regarding AutoML services on SO?

4.4.1.1 Motivation

AutoML is increasingly gaining popularity for domain experts to develop machine learning applications without requiring a comprehensive understanding of the machine learning process. The challenges posed by AutoML tools/-platforms needs to be investigated as a distinct strategy for ML application development. SO is a popular discussion platform for systematically analyzing and studying practitioners' challenges. A detailed analysis of the practitioners' challenges in SO will provide invaluable insight to AutoML researchers and providers into the existing obstacles to AutoML adoption. This analysis will also provide direction for future initiatives to mitigate these obstacles and make machine learning more accessible for various applications and research problems.

4.4.1.2 Approach

We apply LDA topic modeling to our AutoML-related discussion in SO. We get 13 AutoML-related topics, as discussed in Section 4.3. Following previous works [2, 7, 37, 215, 277], we use open card sorting [102] approach to label these AutoML topics. In this approach, there is no predefined list of labels. Following related works [2, 10, 37, 254], we label each topic by analyzing the top 30 words for the topic and a random sample of at least 20 questions from each topic. We assign a label to each topic and discuss it until there is an agreement.

After this initial analysis, i.e., labeling, we merge two topics if they contain similar discussions. For example, we merge topics 1 and 3 into Library/Platform Management because both contain developers' discussions about setting up the development environment and installing packages in a local or cloud environment. Similarly, we merge topics 5 and 13 into Data Management because they contain discussion related to data mining, data storing, and data filtering. In the end, we get 13 different AutoML-related topics.

After this, we reexamined the topic to find any clusters/groups. For example, Data Exploration, Data Management, and Data Transformation topics are related, and thus, they are grouped under the Data topic category. Similarly, we put Model Training & Monitoring, Model Debugging, and Model Design topics under the Model topic category. We use an annotation guideline for labeling the topics and creating the taxonomy of topics to ensure consistency and reproducibility. We share the annotation guide with our replication package.

4.4.1.3 Result

After manually labeling and merging topics, we find 13 AutoML-related topics. Then after grouping these 13 topics into categories, we find four topic categories: (1) MLOps, (2) Model, (3) Data, and (4) Documentation . Figure 4.3 shows the distribution of these four topic categories and their percentage of questions. Among these categories, MLOps has the highest coverage of questions and topics (43% Questions in 5 Topics), followed by the Model category (28% Questions in 4 Topics), Data (27% Questions in 3 Topics), Documentation (2% Questions in 1 Topic).

Figure 4.4 presents the 13 AutoML topics sorted by the number of posts. A post is either an AutoML-related question or an accepted answer. As discussed in Section 4.3.1, our final dataset contains 14,341 posts containing 10,549 questions and 3,792 accepted answers. The topic with the highest number of posts is at the top of the list. On average, each AutoML topic has 1103 posts (#question + #accepted answer). The topic Data Management has the highest number of posts (16.4%), containing 16.8% of total questions and 15.3% total accepted answers. On average, each topic has around 811 AutoML-related SO questions.

Figure 4.5 describes 13 AutoML-related topics into four categories. The topics are presented in descending order of the number of questions. For example, the MLOps category has the highest number of questions, followed by the Model category. Similarly, topics under each category are also organized in descending order of the number of questions. For example, Under the Data topic category, three topics: Data Management (16.8% questions), Data transformation (6.3% questions), and Data exploration (3.9% questions), are organized by the number of questions. We discuss the four AutoML topic categories and the 13 topics below.

MLOps is biggest topic category based on number of questions (43.2% questions). MLOps [178] is a term for Machine Learning Operations, and it focuses on practices to streamline the process of ML engineering from model development to production to monitor their performance over time. This collaborative approach involves stakeholders from

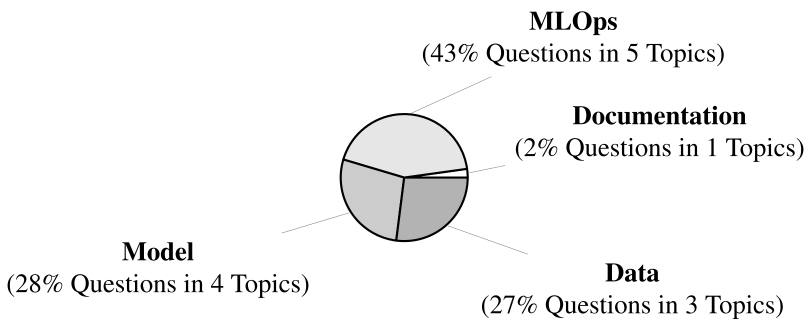


Figure (4.3) Distribution of Questions and Topics per Topic Category.

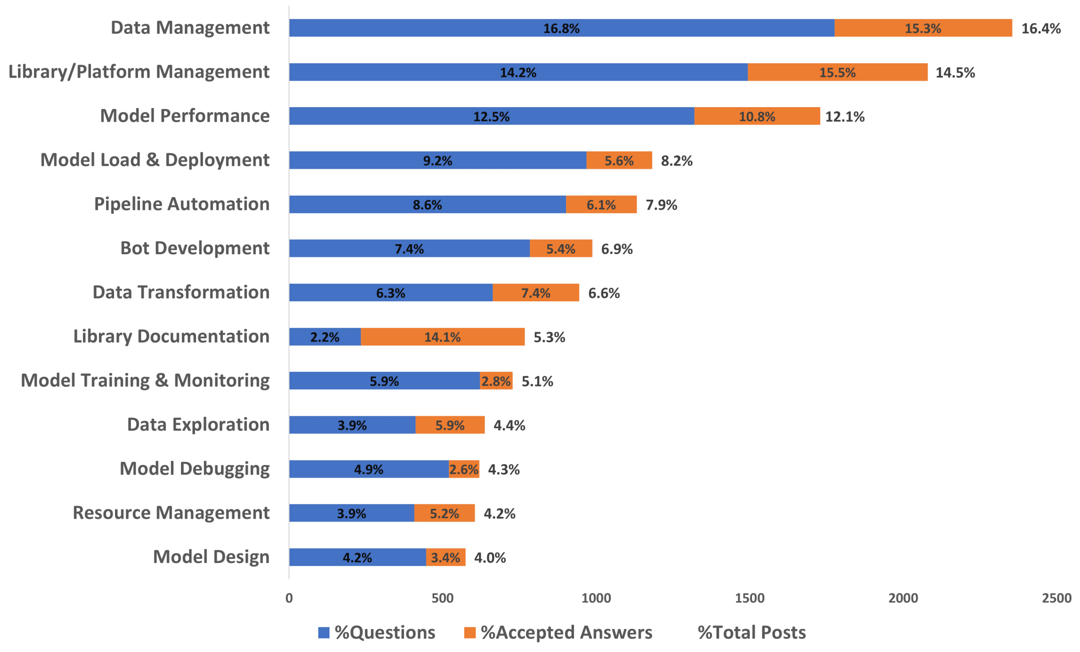


Figure (4.4) Distribution of posts (#Questions + #Accepted Answers) of AutoML topics.

different teams, such as ML engineers and DevOps engineers. This category discusses automating the machine learning pipeline using the AutoML services, setting up local and cloud environments, maximizing resource (i.e., CPU, GPU) management, and deploying the trained model. It has five topics: (1) Topic *Library/Platform Management (14.1%)* contains discussion about setting up a proper development environment, installing ($Q_{51471526}$), upgrading, and downgrading packages ($Q_{70047920}$), resolving incompatible version issues ($Q_{53328131}$, $Q_{71569924}$), portability issues in different operating systems ($Q_{67940893}$) both for local and cloud platforms. For example, in $Q_{56215238}$, a user is inquiring about integrating a third-party TensorFlow library with H2O's flow UI (e.g., "Install new package in H2O Flow UI"). (2) Topic *Model Load & Deployment (9.1%)* topic contains discussion related to training and saving the trained model ($Q_{60733257}$), deploying the trained model via an API endpoint for public use ($Q_{71513161}$, $Q_{69338516}$), how to export a trained host locally ($Q_{56842056}$, $Q_{64467754}$) or different cloud environment ($Q_{54145693}$). For example, in $Q_{71513161}$, a practitioner is asking for help to deploy the newly trained object detection model via AWS Sagemaker public endpoint

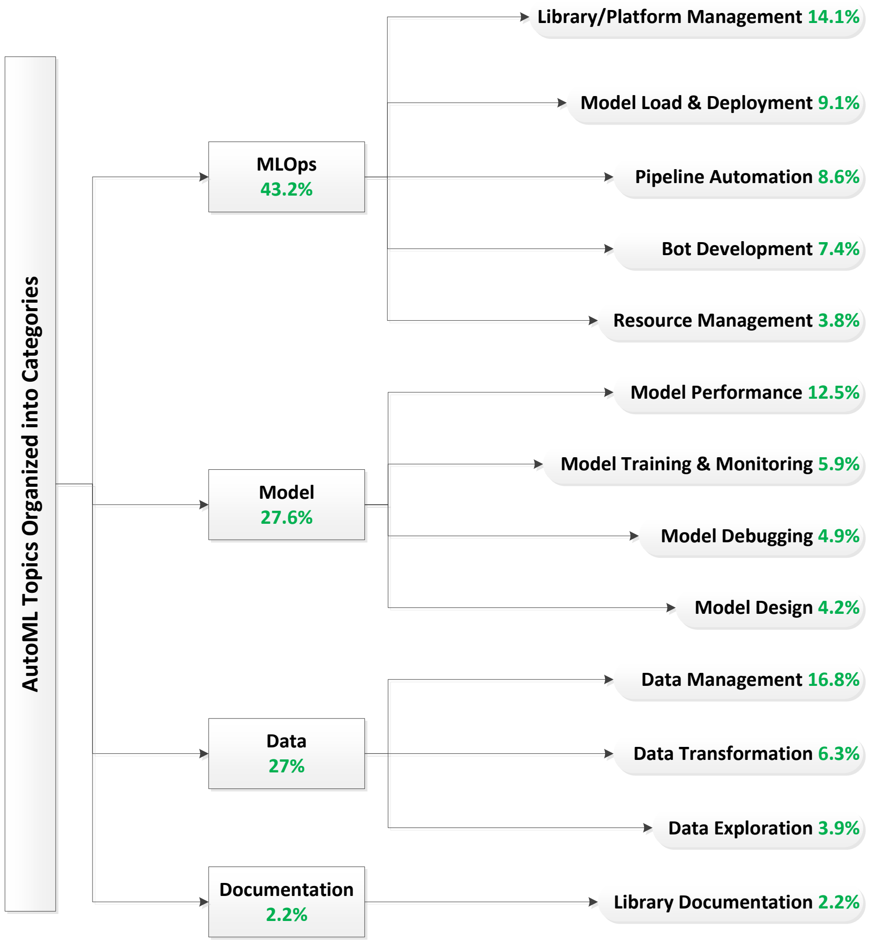


Figure (4.5) A taxonomy of 13 AutoML topics organized into four categories.

(e.g., "How can I create an endpoint for yolov5 inference in AWS sagemaker") (3) Topic *Pipeline Automation (8.6%)* contains discussion about different platform/tool specific features to automating different ML pipelines ($Q_{68115476}$), automatically scheduling experiments ($Q_{30092360}$), custom script to connect and extract data ($Q_{59445324}$), automating data pipeline($Q_{42651900}$), automating CI/CD ($Q_{67945601}$), running profiler ($Q_{61736810}$). For example, this topic contains queries like "How to organize one step after another in [AutoML Platform] Pipelines?" (in $Q_{68115476}$). (4) Topic *Bot Development (7.4%)* contains discussions about how to design, develop and improve the performance of different bots ($Q_{46886079}$), different challenges of developing chatbot ($Q_{55708995}$, $Q_{69490831}$, $Q_{52439304}$), i.e., managing different intents ($Q_{64008462}$), integrating existing bots such as Alexa ($Q_{50993053}$) with new applications, integration of different bots ($Q_{71675543}$), bot deployment ($Q_{71220465}$). So, in this topic, practitioners ask queries such as "Approaches to improve Microsoft ChatBot with each user conversation by learning from it?" in $Q_{46886079}$ or "How to integrate Amazon lex with MS Bot framework?" in $Q_{71675543}$. *We put this topic under MLOps rather than the Model category because after* *analyzing, we find most of the questions of this topic mostly focus on streamlining ML operations by integrating various cloud services, as opposed to the training or debugging the model.* (5) Topic *Resource Management (3.8%)* contains discussions related to resource management for AutoML tools or better configuring the AutoML cloud platform's training setup. It mainly concerns queries regarding how to maximize GPU utilization (e.g., "Can Driverless AI instance utilize K80 GPU for faster performance in GCP?" in$Q_{64651996}$, $Q_{60243228}$), CPU utilization ($Q_{71121820}$, $Q_{36680046}$), using optimal containers/nodes ($Q_{65276017}$), Auto scaling (e.g., "how does scaling policy work with sagemaker endpoints?" in $Q_{71344215}$, $Q_{69904211}$).

Model is the second largest topic category based on the number of questions (27.6% questions). It discusses designing ML models, training, debugging and improving performance. It has four topics: (1) Topic *Model Performance (12.5%)* contains discussion related to improving the AutoML model's performance, i.e., queries related to why one model performs better than another ($Q_{62167244}$), how to improve performance (e.g., "how to improve gradient boosting model fit" in $Q_{61002095}$), compare, monitor and visualize performance metrics (e.g., "How do you monitor more 'standard' metrics with SageMaker?" in $Q_{71898075}$) using the AutoML dashboard. (2) Topic *Model Training & Monitoring (5.9%)* contains discussions about different issues with training the model (e.g., "Why is my Google automl vision trained on cloud much better than the one trained on edge" in $Q_{59896056}$), deleting trained model ($Q_{57750611}$), programmatically training the model ($Q_{60329106}$, $Q_{51873458}$), resources it takes for training (60535298), explainability of the model (e.g., "Explainable AI with Cloud Vision API" in $Q_{72127541}$, $Q_{68251877}$), integrate the model with mobile/web application ($Q_{54781708}$, $Q_{72081132}$), monitor the training progress (e.g., "Vertex AI - how to monitor training progress?" in $Q_{72051655}$). (3) Topic *Model Debugging (4.9%)* contains discussion related to the implementation of different ML algorithms and debugging queries such as importing library ($Q_{37549591}$), error messages (e.g., "H2OAutoML throws libgomp exception in train step" in $Q_{57758996}$, $Q_{66011434}$, $Q_{61803264}$), debugging invalid arguments (e.g., "Invalid argument to unary operator" in $Q_{41058261}$), debugging codes ($Q_{65812334}$). (4) Topic *Model Design (4.2%)* contains discussions on the theoretical discussion of ML experiment design using AutoML services. It contains questions requesting suggestions for the best approach for using AutoML platforms/tools for regression or classification tasks. For example, it contains questions related to machine learning experiment design ($Q_{12651719}$), such as which ML algorithm and metrics to use for various problems (e.g., "Which machine learning algorithm to evaluate the best combinations of groups?" in $Q_{43332966}$, $Q_{37966070}$), feature engineering (e.g., "Questions about feature selection and data engineering when using H2O autoencoder for anomaly detection" in $Q_{64490264}$), time series forecasting ($Q_{54862099}$), unsupervised learning (e.g., "Unsupervised Sentiment Analysis pycaret" in $Q_{63143622}$).

Data is the third largest topic category based on the number of questions (27% questions). It contains practitioners' discussion on data collection and management, data filtering and exploration, and transforming and querying the data. It has three topics: (1) Topic *Data Management (16.8%)* is the biggest topic by the number of questions, and it contains discussion related to the dashboard to configure and manage data mining and security setup (e.g., “Splunk dashboard and reports source backup for versioning” in $Q_{68821885}$, $Q_{61658701}$), automatically sending data for mining (e.g., “How do I send JSON files to Splunk Enterprise from JAVA?” in $Q_{58555219}$, $Q_{23017469}$) and reporting the mining result to another application ($Q_{57149307}$) or via email ($Q_{58204476}$), (2) Topic *Data Transformation (6.3%)* contains discussions related to data encoding, i.e., convert to categorical features ($Q_{50987850}$, $Q_{47513901}$), processing data frames (e.g., “how to join two frames in h2o flow?” in $Q_{37344958}$, $Q_{67452392}$, $Q_{56246048}$), data filtering (e.g., “Filtering h2o dataset by date, but being column imported as time in R” in $Q_{51216030}$), finding patterns using regex ($Q_{10890718}$), read and process large data files (e.g., “How to read large text file stored in S3 from sagemaker jupyter notebook?” in $Q_{65550808}$), processing millions of files ($Q_{62330719}$) (3) Topic *Data Exploration (3.9%)* contains discussions related to creating new attributes in the dataset ($Q_{26195630}$), making/storing/exporting data association rule (e.g., “Make association rules of a certain template” in $Q_{30684638}$, $Q_{52568212}$), data pre-processing ($Q_{25056600}$) and mining (e.g., “Text Mining a single text document” in $Q_{22885133}$, $Q_{24241992}$), issues with making queries on the dataset (e.g., “MongoDB aggregation query in RapidMiner” in $Q_{41068724}$, $Q_{15229636}$),

Documentation is smallest topic category based on number of questions (2.2% questions). It contains a discussion related to various limitations on AutoML API documentation and instructions on using other libraries. It has one topic: (1) Topic *Library Documentation (2.2%)* contains discussion about AutoML tool/platforms API documentation. For example, it contains questions inquiring about resources for details about API parameters (e.g., requesting details on documentation “Python: h2o detailed documentation” in $Q_{60667920}$, $Q_{72385022}$, $Q_{52305760}$), API constraint/unclear API description (e.g., “Could feature interaction constrains be enforced in h2o.xgboost in R” in $Q_{64247245}$), Incorrect API documentation, requesting examples for API usage (e.g., “What are some specific examples of Ensemble Learning?” in $Q_{38135557}$, $Q_{31172775}$).

RQ1. What topics do practitioners discuss regarding AutoML services on SO? We found 13 AutoML-related topics that we organized into four categories. The MLOps category (5 topics) has the highest number of SO questions (43.2%), followed by Model (4 topics, 27.6% questions), Data (3 topics, 27% questions), documentation (1 topic, 2.2% questions). We find that the Data Management topic under the Data category has the highest number of questions (16.8%), followed by Library/Platform Management (14.2%) under MLOps, Model Performance (12.5%) under Model. Our study reveals that AutoML practitioners mostly struggle with integrating different AutoML services to improve model performance and setting up development and deployment environment.

4.4.2 RQ2. How AutoML topics are distributed across different machine learning life cycle phases?

4.4.2.1 Motivation

From our previous analysis, we see that the AutoML topics contain different types of questions. We also observe that these topics are asked about different Machine Learning Life Cycle (MLLC) phases. Machine learning is a distinct domain with its life cycle phases. In order to complete an ML application, a practitioner has to go through an MLLC phase such as problem defining, data collection and processing, model development, deployment, and monitoring. For example, API documentation-related questions are mostly asked during the Model Training phase. Therefore an in-depth analysis of questions asked during the six MLLC phases may offer us a better understanding of the discussed AutoML challenges. In this research question, we aim to investigate which ML phases are most challenging and how these topics are distributed across these MLLC phases. This analysis may provide AutoML practitioners with a better understanding of ML application development using AutoML tools/platforms. This analysis will provide valuable insights for AutoML researchers and tools/platform providers.

4.4.2.2 Approach

In order to better understand how the questions are discussed in SO on MLLC, we take a statistically significant sample from our extracted dataset and then manually analyze the questions and label them into one of MLLC phases. So, our approach is divided into two main steps: Step 1. We generate a statistically significant sample size from our dataset, Step 2. we manually analyze and label those questions.

Step 1. Generate Sample. As we discussed in Section 4.3.1 our dataset has 10,549 SO questions. A statistically significant sample with a 95% confidence level and five confidence intervals would be at least 371 random questions, and a 10 confidence interval would have a sample size of 95 questions. A random sample is representative of the entire dataset, but it could miss questions belonging to smaller topic categories. For example, as discussed in RQ1, we have 13 AutoML topics organized into four categories. As random sampling from the dataset is not uniform across the topic categories, it might miss important questions from smaller topic categories such as Documentation. Therefore, following previous empirical studies [2, 254], we extract a statistically significant random sample question from each of the four topic categories. More specifically, we extract SO questions in our sample from each of the four topic categories with a 95% confidence level and 10 confidence intervals. The sample dataset is drawn as follows: 94 questions from the MLOps topic category (total question 4,557), 93 questions from the Model topic category (total question 2,907), 93 questions from the Data topic category (total question 2,851), 68 questions from Documentation topic category (total question 234). In summary, we sample a total of 348 questions.

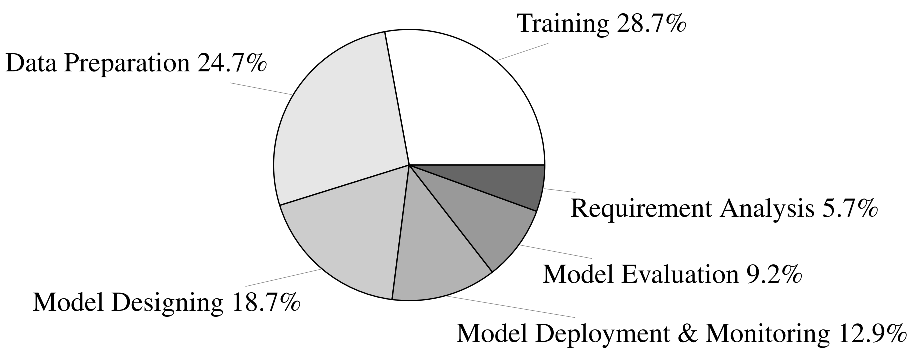


Figure (4.6) Distribution of questions (Q) per MLLC phase

Table (4.1) Distribution (frequency) of AutoML topics per MLLC phase. Each colored bar denotes a phase (Black = Requirement Analysis, Green = Data Preparation, Magenta = Model Designing, Red = Training, Blue = Model Evaluation, Orange = Model Deployment & Monitoring)

Topics	**MLLC Phases Found in #Questions**					
MLOps (94)	⬛ 4	🟩 9	🟪 10	🟥 37	🟦 8	🟧 26
Model (93)	⬛ 7	🟩 9	🟪 27	🟥 24	🟦 18	🟧 8
Data (93)	⬛ 2	🟩 59	🟪 11	🟥 11	🟦 2	🟧 8
Documentation (68)	⬛ 7	🟩 10	🟪 17	🟥 27	🟦 4	🟧 3

Step 2. Labeling MLLC phase. We analyze and label questions from our samples into the following six phases. Following similar work by Alshangiti et al. [14] and others [9, 134], we label each question into one of the following six phases. (1) Requirement Analysis, (2) Data Preparation, (3) Model Designing, (4) Training, (5) Model Evaluation, (6) Model Deployment & Monitoring.

4.4.2.3 Result

Figure 4.6 presents the distribution of our AutoML questions into six MLLC phases. We find that the Model training, i.e., the implementation and debugging phase, has 28.7% of our 348 annotated questions, followed by Data Preparation (24.7%), Model Designing (18.7%), Model Deployment & Monitoring (12.9%), Model Evaluation (9.2%), and Requirement Analysis (5.7%). SO is a technical Q&A platform, and developers/practitioners use it for solving technical issues; thus, we find many debugging and troubleshooting related queries in Model implementation and data pre-processing related. However, we also find that AutoML practitioners ask many queries related to designing (e.g., $Q_{56086214}$) and deploying AutoML applications (e.g., $Q_{69113428}$). This analysis highlights that AutoML practitioners struggle in using AutoML services. We provide an overview of the types of questions asked during these six MLLC phases.

Requirement Analysis (20, 5.7%). This is the first stage of any ML project. In this phase, the scope and definition of the problem are defined based on the project's goal. There are questions on the AutoML tools/platforms' features (e.g., "Does AzureML RL support PyTorch?" in $Q_{64025308}$), architectural design of the platform (e.g., "What is Splunk architecture?” in $Q_{28186144}$, comparison of different AutoML services (e.g., “What are the differences between AWS sagemaker and sagemaker pyspark?” in $Q_{66817781}$), condition for their usage (e.g., “Is it possible to share compute instance with other user?” in $Q_{60539094}$).

Data Preparation (86, 24.7%). Based on the problem definition, necessary data is collected from many different sources. The performance of the ML model largely depends on the quality of the data. So proper data filtering (e.g., $Q_{55753324}$), pre-processing, formatting, visualization (e.g., in $Q_{62188204}$), and transformation is very important. So, in this phase, we find questions related to large data processing (e.g., “Reading a large csv from a S3 bucket using python pandas in AWS Sagemaker” in $Q_{48111034}$).

Model Designing (65, 18.7%). At this step, a supervised or unsupervised learning approach is applied to the collected dataset for various tasks such as regression, classification, clustering, and forecasting. At this step, the optimal set of features from the dataset is chosen for the ML algorithms. This crucial step determines the performance of the prediction of the model. So, in this phase, we see discussion related to different theoretical discussions of different approaches for the ML model (e.g., in $Q_{26410001}$). For example, in $Q_{15342081}$, a practitioner is querying about how to use ‘:)’ and ‘:(’ emoticons to classify the sentiment of sentences (e.g., “sentiment analysis for arabic language”). We can also see queries regarding the implementation of custom algorithms in AutoML platforms (e.g., in $Q_{56086214}$)

Model Training (100, 28.7%). At this stage, the implementation of the ML model and training takes place. At this step, the model learns from data and gets better at prediction. So, in this phase, we find many queries related to different issues of model training (e.g., “Cannot run huge Python program” in $Q_{35237226}$), debugging different issues, using AutoML cloud platforms UI for training (e.g., “Amazon Lex slot won’t save value in it” in $Q_{64244554}$), etc.

Model Evaluation (32, 9.2%). At this stage, the performance of the trained model is evaluated on unseen data and reported using different metrics. If the output is not desirable, then the previous steps are iterated. So, in this phase, we find queries related to the prediction of the trained model. For example, in $Q_{57858737}$, a practitioner is querying about different predictions with the same model in the cloud vs. the local environment. Practitioners also ask questions related to error while making a prediction (e.g., “Error in a multiclass classification in during of a prediction (Pycaret Predict function)” in $Q_{70956772}$).

Model Deployment & Monitoring (45, 12.9%). This is the last stage of an ML application. The trained model is deployed into the production environment for general usage at this phase. Before deploying the model, the robustness and scalability need to be considered. The deployed model must be consistently monitored to ensure its prediction on the unseen dataset. At this step, we find queries related to deploying models in the AutoML cloud environment (e.g., “Model deployment in Azure ML” in $Q_{69113428}$), monitoring the model’s prediction (e.g., $Q_{71471108}$), exporting the trained model from the cloud environment (e.g., $Q_{56952741}$).

Topic Categories in different MLLC phases. We find that for all four topic categories, AutoML practitioners need some community support from requirement analysis to implementation to deployment (e.g., "Deploying multiple huggingface model through docker on EC2" in $Q_{71187430}$). We analyze and present how AutoML topics are distributed across six MLLC phases. Table 4.1 presents the distribution of six MLLC phases for each topic category. Our analysis shows that the MLOps topic Category, which encapsulates overall machine learning operations from model development to deployment, contains questions that are asked during the Model development and training (39%) and Deployment (28%) phases regarding various MLOps. Similarly, most questions from Model topic categories are dominant during the model design (29%) and model training phase (26%) regarding how to best design and train the ML model. Around 63% questions from the Data topic category are asked during the data preparation and pre-processing phase regarding data filtering and data querying. We also find that the Document topic category is mostly asked during model training (25%) and model designing phase (40%), highlighting the different limitations of API documentation.

RQ2. How are AutoML topics distributed across different machine learning life cycle phases? We manually annotate 348 questions into one of six AutoML life cycle phases by taking statistically significant sample questions from each of the four topic categories. We find that Model training, i.e., the implementation phase is the most dominant with 28.7% questions, followed by Data Preparation (24.7% questions), Model Designing (18.7% questions), Model Deployment & Monitoring (12.9% questions), Model Evaluation (9.2% questions), and Requirement Analysis, i.e., scope definition (5.7% questions). We find that during the Implementation phase troubleshooting, an error message or stack traces-related questions dominate. However, during the Requirement Analysis, Model Design, and Model Deployment phases, most queries are related to different features or limitations of AutoML services. So better tutorials and learning resources may help the AutoML practitioners in the early stages of the ML Pipeline.

4.4.3 RQ3. What AutoML topics are most popular and difficult on SO?

4.4.3.1 Motivation

Our previous analysis shows that AutoML practitioners face many challenges related to machine learning development and specific challenges to AutoML tools/platforms (e.g., MLOps, Model development, and debugging). Some of these posts are more common than others, i.e., more community participation (i.e., view counts, up-votes, etc.). So, challenges faced for a particular topic during a specific MLLC phase are not equally difficult to get a working solution. A thorough analysis of the popularity and difficulty of practitioners' challenges may yield valuable insight. This analysis may help the researchers and AutoML providers to prioritize efforts to take necessary actions on the usability and applicability of these tools/platforms to make to more accessible to practitioners, especially domain experts.

4.4.3.2 Approach

In this research, we use the following five attributes from our SO dataset to determine the popularity or difficulty of a group of questions. We compute the difficulty for a group of questions using two attributes for each of the questions in that group (1) The percentage of questions that do not have an accepted answer, (2) Average median time to get a solution. Similarly, we estimate how popular the topic is throughout the SO community using the following three popularity metrics: (1) Average #views, (2) Average #favorites, (3) Average score.

These five metrics are common attributes of a SO question, and numerous research in the field [2, 7, 10, 37, 254] have utilized them to assess a question's popularity and the difficulty of finding a solution. One question may have several answers on SO, and the user who asked the question can mark one reply as correct. As a result, the accepted response is regarded as accurate or of high quality. Therefore, the lack of an accepted response can mean the user could not find a suitable, useful response. The proper problem description, i.e., the quality of the question, helps to get an accepted answer. However, the main reason for not having an accepted answer is most likely because the SO community find that problem, i.e., the question difficult. A crowd-sourced platform like SO depends on its users' ability to supply accurate, usable information swiftly. In SO, the median time to receive an answer is only about 21 minutes [254], but questions from a challenging or domain-specific topic may require more time to be answered.

It might be challenging to evaluate the topics' popularity and the difficulty of obtaining an accepted answer using a variety of measures. As a result, following related study [254], we calculate two fused metrics for topic popularity and difficulty. The two fused metrics are described below.

Fused Popularity Metrics. We begin by calculating the popularity metrics for each of the thirteen AutoML topics. However, the range of these attributes varies a lot. For example, the average number of views is in the range of hundreds, average scores may range from zero to five, and the average number of favorites might range from zero to three. Therefore, following related study [254], we normalize the values of the metrics by dividing them by the mean metric value across all groups (e.g., for topics $K = 13$). In this way, we get their normalized popularity metrics: $View\_N_i$, $Favorite\_N_i$, and $Score\_N_i$ for each topic i (e.g., AutoML topics K=13). Finally, following related work [254], we calculate the average fused popularity metric $Fused\_Popularity_i$ for each topic.

$$View\_N_i = \frac{View_i}{\frac{\sum_{j=1}^{K} View_j}{K}} \tag{4.1}$$

$$Favorite\_N_i = \frac{Favorite_i}{\frac{\sum_{j=1}^{K} Favorite_j}{K}} \tag{4.2}$$

$$Score\_N_i = \frac{Score_i}{\frac{\sum_{j=1}^{K} Score_j}{K}} \tag{4.3}$$

$$Fused\_Popularity_i = \frac{View\_N_i + Favorite\_N_i + Score\_N_i}{3} \tag{4.4}$$

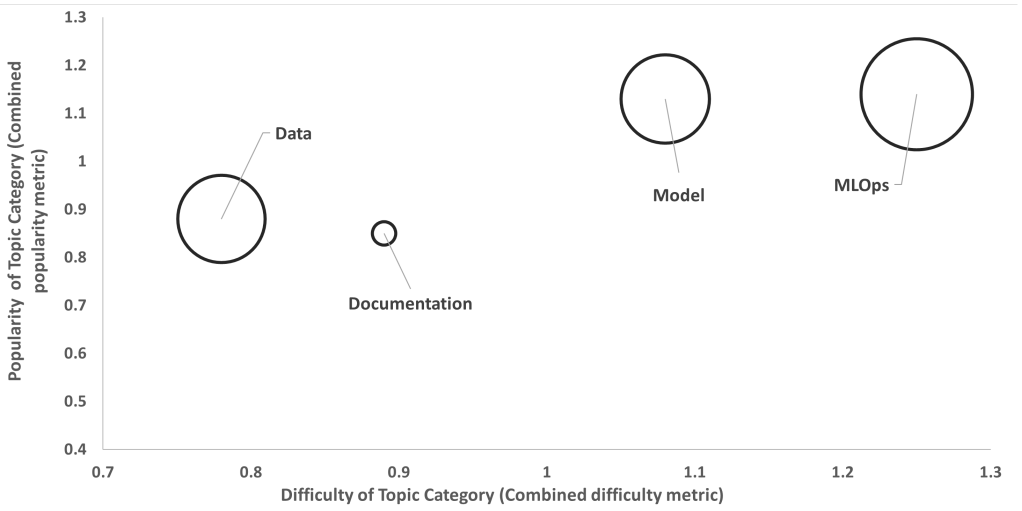


Figure (4.7) The popularity vs. difficulty of AutoML topic categories.

Similar to popularity measurements, we begin by calculating the complexity metrics for each issue. Then, the metric values are normalized by dividing them by the group-wide average metric value (e.g., 40 for LCSD topics). Consequently, we generate two new normalized measures for a given topic i. Finally, the fused difficulty metric $FusedDi$ of topic i is calculated by averaging the normalized metric values.

Fused Difficulty Metrics. Similar to topic popularity metrics, we begin by calculating the difficulty metrics for each topic. Then, we compute the normalized difficulty metrics for a given topic i by dividing them by the mean value across all groups (e.g., 13 topics). Finally, We compute the fused difficulty metric $Fused\_difficulty_i$ of topic i by calculating the average of these two normalized metrics.

$$PctQuesWOAccAns\_N_i = \frac{PctQWoAcceptedAnswer_i}{\frac{\sum_{j=1}^{K} PctQWoAcceptedAnswer_j}{K}} \tag{4.5}$$

$$MedHrsToGetAccAns\_N_i = \frac{MedHrsToGetAccAns_i}{\frac{\sum_{j=1}^{K} MedHrsToGetAccAns_j}{K}} \tag{4.6}$$

$$Fused\_difficulty_i = \frac{PctQuesWOAccAns\_N_i + MedHrsToGetAccAns\_N_i}{2} \tag{4.7}$$

Additionally, we aim to establish a relationship between the difficulty and popularity of these topics. In this study, we utilize the Kendall Tau correlation [141] to determine the relationship between the popularity and difficulty of a topic. In contrast to the Mann-Whitney correlation [148], it is not subject to outliers in the data. We can not analyze the popularity and difficulty of these topics because SO does not provide times data for these popularity metrics [2,10,254].

Table (4.2) Popularity metrics for AutoML topics.

Topic	Category	Fused_Popularity	#View	#Favorite	#Score
Model Performance	Model	4.48	892.4	0.3	1.2
Model Load & Deployment	MLOps	4.05	665.7	0.3	1.1
Library/Platform Management	MLOps	3.89	983.5	0.2	1.1
Data Transformation	Data	3.87	1054.9	0.2	1
Resource Management	MLOps	3.26	625	0.2	1
Pipeline Automation	MLOps	3.11	513.9	0.2	1
Model Design	Model	2.8	546.9	0.2	0.7
Model Debugging	Model	2.77	786.7	0.1	0.9
Data Management	Data	2.67	1047.8	0.1	0.5
Library Documentation	Documentation	2.61	672.2	0.1	0.9
Bot Development	MLOps	2.51	602.5	0.1	0.9
Data Exploration	Data	2.02	671.4	0.1	0.4
Model Training & Monitoring	Model	1.96	376.9	0.1	0.7

Table (4.3) Difficulty for getting an accepted answer for AutoML topics

Topic	Category	Fused_difficulty	Med. Hrs To Acc.	Ques. W/O Acc.
Model Load & Deployment	MLOps	1.75	45.9	70
Model Training & Monitoring	Model	1.4	32	72
Pipeline Automation	MLOps	1.09	22.9	63
Model Design	Model	1.05	21.3	63
Bot Development	MLOps	1.04	19.2	69
Resource Management	MLOps	0.97	17.4	65
Data Transformation	Data	0.96	17.9	63
Library/Platform Management	MLOps	0.9	15.9	62
Model Debugging	Model	0.88	12.6	70
Model Performance	Model	0.79	12.1	60
Data Exploration	Data	0.77	11.6	60
Library Documentation	Documentation	0.76	14.2	49
Data Management	Data	0.63	5	63

4.4.3.3 Result

In Figure 4.7, we present an overview of the popularity vs. difficulty of our four AutoML topic categories. The bubble size represents the number of questions, i.e., the bigger the bubble, the more questions the topic category has. It shows that MLOps is the most difficult topic category, followed by Models, Documentation, and Data. Similarly, MLOps is the most popular topic category, followed by Model, Data, and Documentation. Our observation suggests that AutoML practitioners find various aspects of developing ML models via AutoML tools/platforms popular as well as difficult related to improving the performance of the ML model (e.g., $Q_{37196368}$), deploying the trained model via API endpoint (e.g., $Q_{50334563}$). Similarly, we find that questions related to AutoML SO community have strong support for finding relevant API parameters via official documentation (e.g., $Q_{60667920}$) but relatively less support on the operational and management of these models.

Topic Popularity. Table 4.2 presents the three popularity metrics: Average number of 1. Views, 2. Favorites, 3.

Table (4.4) Correlation between the topic popularity and difficulty.

coefficient/p-value	**View**	**Favorites**	**Score**
% Ques. w/o acc. ans	-0.24/0.26	-0.08/0.73	-0.07/0.75
Med. Hrs to acc. ans	-0.51/0.01	0.28/0.23	0.14/0.53

Scores. It also contains the combined popularity metrics (i.e., Fused_Popularity) using Equation 4.4. In the Table, the AutoML topics are presented in descending order based on the Fused_Popularity metric. Model Performance topic from the Model topic category has the highest Fused_Popularity score(i.e., 4.9), highest average favorite count (i.e., 0.3), and average score (i.e., 1.2). This is the third largest topic (e.g., 12.5% of questions), and it reflects the common struggle of AutoML practitioners to enhance the model's performance using the best AutoML configuration (e.g., in $Q_{59219818}$). The Data Transformation topic under the Data category has the highest average view count (i.e., 1055), followed by Data Management (i.e., 1028) and Library/Platform Management (i.e., 984) from MLOps. Data filtering and prepossessing are crucial for optimal model performance, and our analysis reveals that AutoML practitioners regularly face these challenges. For example, in $Q_{38927230}$, a practitioner is struggling to text pre-processing using the Pandas library on the Azure cloud platform, which has around 45K views. Model Training & Monitoring from the Model category has the lowest Fused_Popularity score(i.e., 1.96) and average favorite count (i.e., 0.1). This topic contains questions that are tool/platform-specific (e.g., $Q_{51878538}$, $Q_{58177571}$), and so attract a small number of AutoML practitioners.

Topic Difficulty. In Table 4.3, we present the summary of two difficulty metrics: 1. Percentage of questions without accepted answers, 2. Median hours to get accepted answers for our 13 AutoML topics. Similar to topic popularity, we also report the combined topic difficulty metrics (e.g., Fused_difficulty) using Equation 4.7. The AutoML topics in Table 4.3 are organized in descending order based on the Fused_difficulty value. Topic Model Load & Deployment is the most difficult topic in terms of Fused_difficulty metric (i.e., 1.75), median hours to get accepted answers (i.e., 45.9 hours), the second highest number of questions without accepted answers (i.e., 70%). Model Training & Monitoring topic from the Model category has the highest number of questions without any accepted answers (i.e., 72%) followed by Model Load & Deployment from MLOps category with (i.e., 70%) and Model debugging (i.e., 70%). These questions receive less help from the SO community since they are highly specific to a particular tool or platform (e.g., $Q_{58320938}$) and are difficult to reproduce for others. Thus they frequently lack accepted answers (e.g., $Q_{65921153}$, $Q_{70950914}$). Library Documentation has the second least difficult topic in terms of Fused_difficulty (i.e., 0.76) with the least amount of questions without accepted answers (i.e., 49%). Frequently, AutoML practitioners lack the abilities required to discover relevant documentation, and the AutoML SO community is quite supportive in providing the required resources (e.g., $Q_{40227519}$, $Q_{42446046}$). Data Management under the Data category is the least difficult topic with a Fused_difficulty value of 0.63, the least median hours to get accepted answers (i.e., 5 hours). These queries are typical for data analytics (e.g., $Q_{54896417}$) and some pre-processing data tasks similar to traditional ML application development (e.g., $Q_{24700708}$), therefore a border ML community combined with AutoML practitioners can contribute accepted answers.

Correlation between topic Popularity vs. Difficulty. We systematically explore if there is any positive or negative co-relationship between topic popularity and difficulty. For example, we find that Model Load & Deployment topic is the most difficult and the second most popular topic. Alternatively, we find that Model Training & Monitoring is the least popular but the second most difficult topic.

Table 4.4 presents the six correlation measures between topic fused difficulty and popularity metric presented in Table 4.2 and 4.3. Four out of six correlation coefficients are negative, and the other two are positive; thus, they are not statistically significant, with a 95% confidence level. Therefore, from this analysis, we can not say the most popular topic is the least difficult to get an accepted answer to and vice versa. Nonetheless, AutoML researchers and platform providers could use this insight to take further necessary steps to accept solutions to most of the popular AutoML topics.

RQ3. What AutoML topics are most popular and difficult on SO? MLOps is the most challenging category, followed by Model, Documentation, and Data. We find that model performance improvement-related topics and operational aspects of ML pipeline, i.e., the configuration of library and packages and model deployment-related topics, are most common and popular among AutoML practitioners. Similarly, we also find that AutoML practitioners find Automated ML pipeline, model deployment & monitoring related topics most difficult. In summary, we find that the topics that are more specific to AutoML services, i.e., have relatively greater difficulty than those that are generic to all tools/platforms or ML development.

4.4.4 RQ4. How does the topic distribution between cloud and non-cloud AutoML services differ?

4.4.4.1 Motivation

Both cloud-based and non-cloud-based AutoML services are present in our dataset. Each service provides a unique collection of services; their use cases are distinct. Cloud-based solutions, for instance, emphasize an end-to-end pipeline for data collecting to model operationalization, whereas non-cloud-based services offer greater customization for high-level data preparation, feature engineering, model evaluation, and model updating. In this study, we investigate how AutoML-related topics, MLLC life cycle phases, and topic popularity and complexity vary between cloud-based and non-cloud-based AutoML solutions. This analysis will provide valuable insights into the challenges of cloud versus non-cloud low-code (i.e., AutoML) services and opportunities for further development of these solutions.

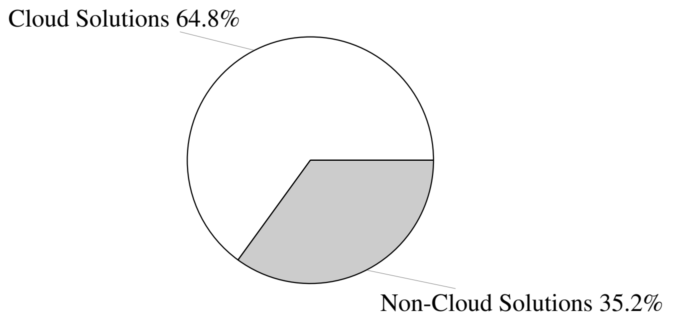


Figure (4.8) Distribution of questions on cloud vs non-cloud AutoML services.

4.4.4.2 Approach

In order to understand how the challenges of cloud vs. non-cloud AutoML solutions differ, first, we need to separate cloud vs. non-cloud AutoML SO questions. So our approach is divided into two main steps: (1) Step 1. Identify SO tags for cloud vs. non-cloud solutions, (2) Step 2. Separating SO questions,

Step 1. SO tags separation. In order to accomplish this, we manually analyze the 41 AutoML-related tags, as explained in Section 4.3, to determine which SO tags correspond to cloud-based or non-cloud-based AutoML solutions. For instance, we classify the "amazon-sagemaker" SO tag as a cloud solution because it contains discussions about the Amazon cloud SageMaker solution, whereas the "pycaret" SO tag is classified as a non-cloud solution because it contains discussions about the python-based, non-cloud AutoML framework PyCaret.

Step 2. SO questions separation. We must separate SO questions; however, a SO question may contain between one to five SO tags. We consider a query to belong to the non-cloud category if it has both cloud and non-cloud tags, as this provides us with greater granularity. For example, (e.g., "How can I automate Splunk iterations using REST API" in $Q_{48421583}$) contains SO tags "splunk", "splunk-sdk" which cloud and non-cloud tags respectively. Considering these queries as non-cloud (i.e., AutoML tools/framework) serves our objective more effectively.

After this, we analyze how AutoML topic categories, MLLC phases, and topic popularity and difficulty differ in the cloud vs. non-cloud solutions. For cloud vs. non-cloud distribution of MLLC phases, we use the same 348 annotated SO questions described in Section 4.4.2.2.

4.4.4.3 Result

Figure 4.8 offers an overview of the cloud versus non-cloud AutoML service SO question in our dataset. After manual analysis, we find 23 SO tags out of 41 are related to cloud-based solutions, while the remaining 18 are related to non-cloud-based solutions. We observe that 6,833 (i.e., 64.8%) SO questions belong to the cloud, and 3,716 (i.e., 35.2%) SO questions belong to non-cloud AutoML solutions. This insight demonstrates the prevalence and popularity of cloud-based AutoML services.

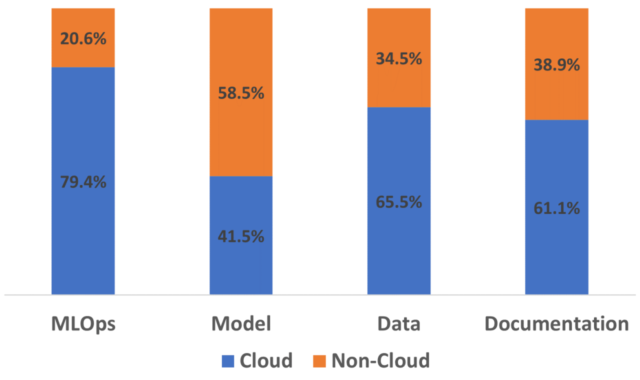


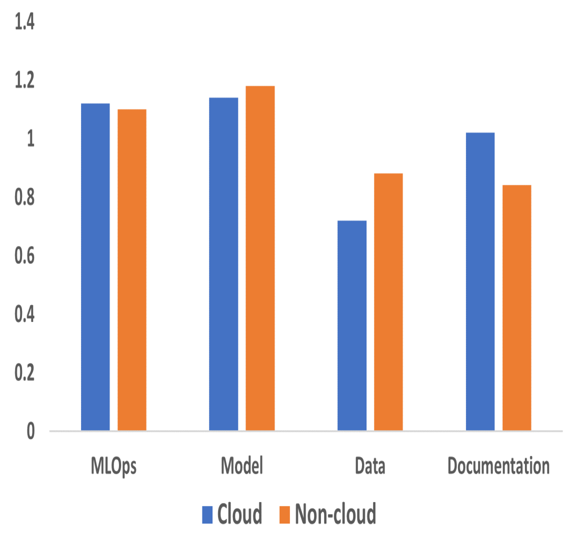
Figure (4.9) The distribution of AutoML topic categories between cloud vs non-cloud AutoML services.

Table (4.5) Distribution (frequency) of MLLC phase between cloud vs non-cloud AutoML services. Each colored bar denotes a phase (Black = Cloud, Green = Non-cloud Solution)

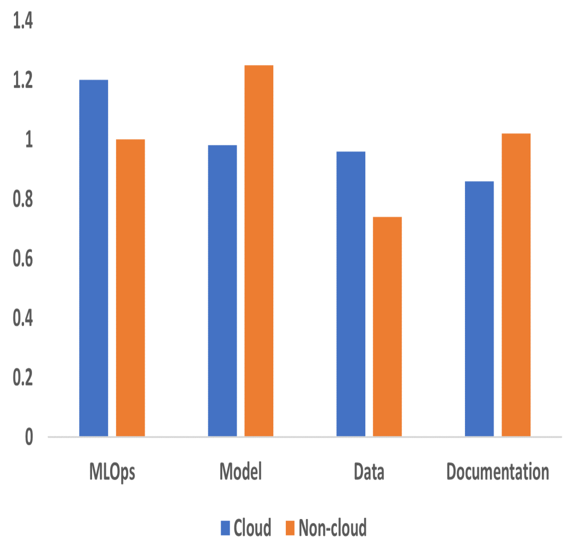
MLLC Phases	#Questions in cloud vs non-cloud solution
Requirement Analysis	▬ 13 ▬ 7
Data Preparation	▬ 60 ▬ 27
Model Designing(MD)	▬ 36 ▬ 29
Model Training (MT)	▬ 54 ▬ 45
Model Evaluation (ME)	▬ 12 ▬ 20
Model Deployment and Monitoring (MDM)	▬ 36 ▬ 9

Figure 4.9 shows the distribution of AutoML topic categories between cloud vs. non-cloud services. We can see that for the MLOps category, most of the questions (i.e., 80%) belong to cloud platforms. Many of these questions belong to model deployment (e.g., "How can I deploy an ML model from ECS to Sagemaker?" in $Q_{48537811}$) and (e.g., "Monitoring the performance of ML model on EC2 Instance" in $Q_{66655967}$). Similarly, we can see that for the Data category, cloud-based solutions dominate (66%) with queries related to cloud data management (e.g., "Load Azure ML experiment run information from datastore" in $Q_{66801546}$). Non-cloud-based services dominate (59%) on Model related queries (e.g., "Retraining a model in PyCaret" in $Q_{68330216}$), (e.g., "Autokeras training model 10x slower (Google Colab)" in $Q_{68878481}$). AutoML cloud solutions also have similar queries related to the Model's training performance (e.g., "Training on Sagemaker GPU is too slow" in $Q_{67213383}$). For the Documentation category, around 62% of questions belong to cloud-based AutoML services where practitioners ask queries like (e.g., "API call for deploying and undeploying Google AutoML natural language model - documentation error?" in $Q_{67470364}$). Non-cloud-based services practitioners ask questions requesting detailed documentation in $Q_{60667920}$.

Table 4.5 demonstrate the distribution of cloud vs. non-cloud solution across different MLLC phases. Among these 348 SO questions, 211 (i.e., 60%) questions belong to AutoML cloud solutions and 137 (i.e., 40%) questions belong to non-cloud AutoML solutions. We observe that practitioners using AutoML cloud solutions dominate queries during the Model Deployment and Monitoring (82%) Phase, (e.g., "how to deploy the custom model in amazon sageMaker" in



(a) A comparison of difficulty between cloud vs non-cloud topic categories



(b) A comparison of popularity between cloud vs non-cloud topic categories

Figure (4.10) An overview of combined difficulty and popularity between cloud vs non-cloud AutoML solution's topic categories.

$Q_{61248115}$, deployment failure issue in $Q_{61683506}$). We also observe that during the Model Evaluation phase, non-cloud AutoML solution-related queries dominate (63%) (e.g., "[H2O] How do I evaluate a multinomial classification model in R?" in $Q_{43556802}$, "Evaluate AutoKeras model gives different results then the same model as pure Keras evaluate (h5 file)" in $Q_{54491965}$). During other MLLC phases, the proportion of questions about cloud versus non-cloud solutions remains almost similar to their original distribution (e.g., Requirement Analysis (65% vs. 35%), Data Preparation (57% vs. 43%) and Model Training phase (55% vs. 45%), Model Designing (MD) phase (55% vs. 45%).

Figure 4.10 presents the comparison of popularity and difficulty (Section 4.4.3) of topic categories of AutoML cloud vs. non-cloud solutions. For the Documentation subject category AutoML, cloud-related inquiries are slightly more challenging (fused difficulty metric 1.02 vs. 0.84); however, for the Data topic category AutoML, non-cloud-related queries are more challenging (i.e., fused difficulty metric 0.88 vs. 0.72). For MLOps and Model, cloud and non-cloud-related queries are almost equally challenging. According to the fused popularity metric, AutoML cloud solution-related queries on MLOps (i.e., 1.2 vs. 1.0) and Data (i.e., 0.96 vs. 0.74) topic categories are somewhat more popular. In contrast, AutoML non-cloud related SO queries on Model (i.e., 1.25 vs. 0.98) and Documentation (i.e., 1.02 vs. 0.86) topic categories are more popular.

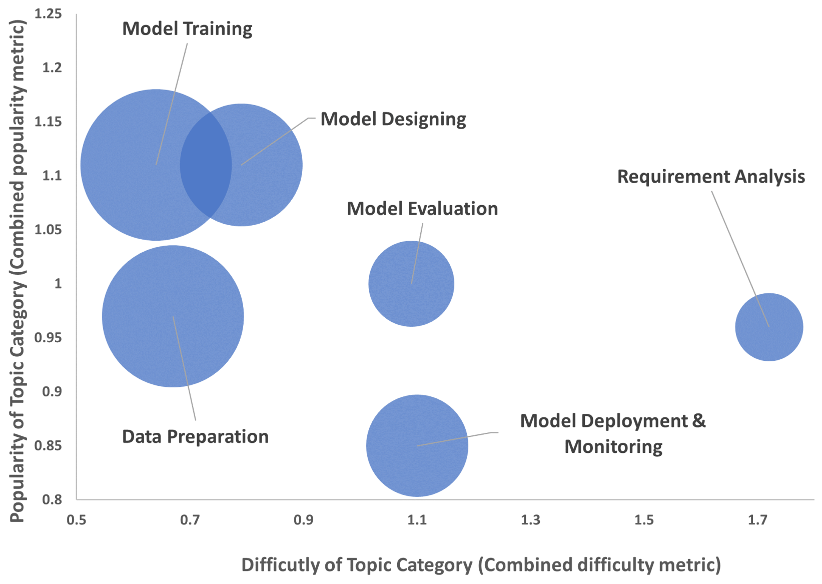


Figure (4.11) The popularity vs. difficulty for AutoML pipeline phases.

RQ4. How does the topic distribution between cloud and non-cloud AutoML services differ? In our dataset, around 64.8% SO questions belong to the cloud, and 35.2% belong to non-cloud-based AutoML solutions. MLOps (i.e., 80%), Data (i.e., 66%), and Documentation (i.e., 61%) topic categories are dominated by cloud-based AutoML solutions. They are comparable in terms of difficulty but marginally more popular, except for the Documentation category, which is marginally more challenging and less popular. Alternatively, the Model (59%) topic category is predominant and popular across non-Cloud-based AutoML solutions. For the Model Deployment and Monitoring phase, AutoML cloud solutions-related questions predominate (82%). However, during the Model Evaluation phase, non-cloud AutoML solution-related queries predominate (63%).

4.5 Discussions

4.5.1 Evolution of AutoML related discussion

Figure 4.13 depicts the progression of overall AutoML-related discussion from our extracted dataset between 2008 to 2022. Additionally, it demonstrates that AutoML-related discussion is gaining popularity in mid-2016 (i.e., more than 200 questions per quarter). Figure 4.13 shows that during the first quarter of 2021, the total number of AutoML questions on Stack Overflow experiences a considerable decrease (i.e., about 25%), primarily due to the low number of queries in Splunk and Amazon Sagemaker during the pandemic. However, Amazon launches a suite of new AutoML services at the start of 2021, and the overall trend for AutoML-related discussion shows an upward trend [34]. We

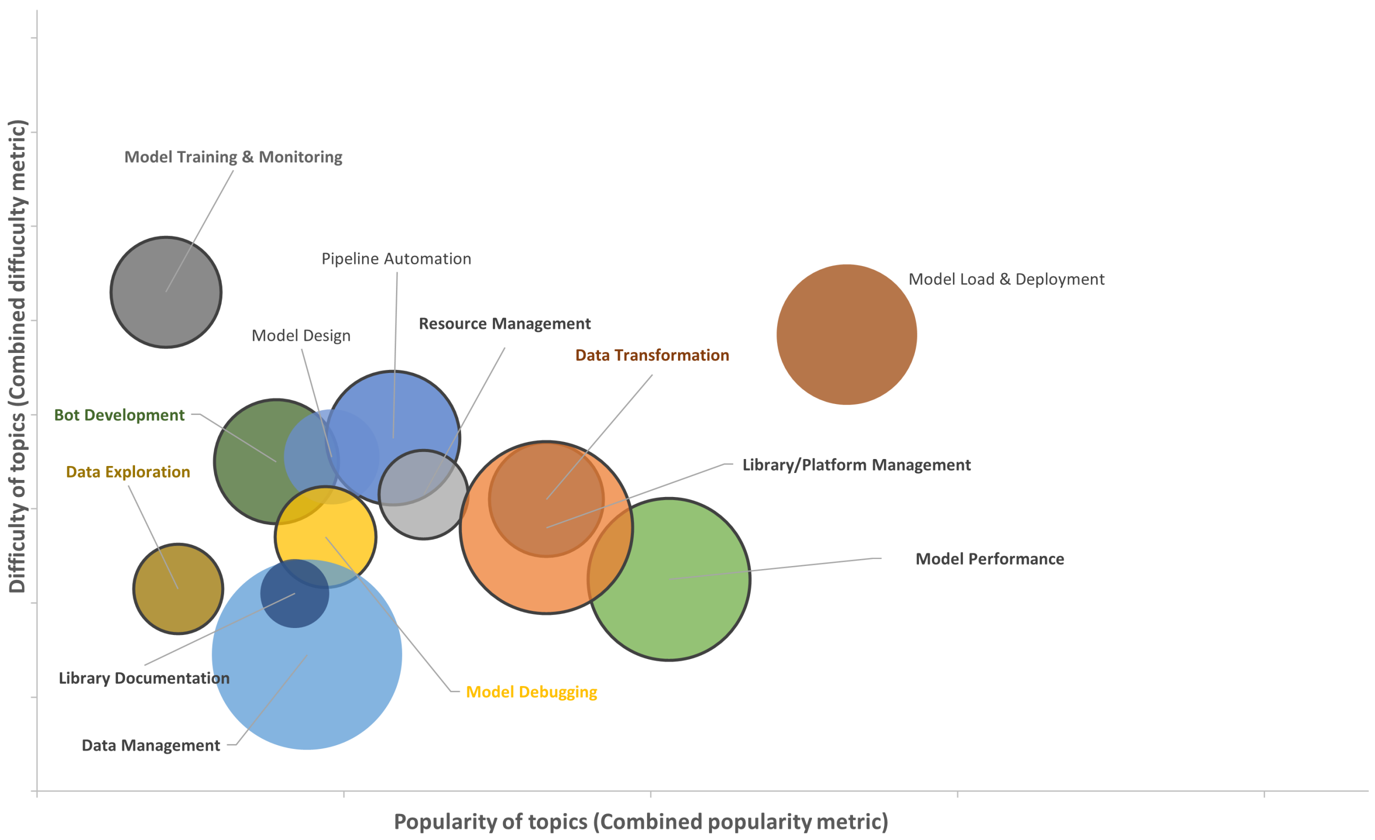


Figure (4.12) The popularity vs. difficulty for AutoML topics.

provide a more detailed explanation for Figure 4.13 below.

MLOps This is the largest AutoML topic category, with around 43% of AutoML-related questions and five AutoML topics. From Figure 4.14, we can see this topic category began to gain popularity from the start of 2017, i.e., after big tech companies started to offer AutoML cloud platform services (Section 4.2.2). From 2017 to 2022, AutoML library and cloud platform management-related topics remain the most popular in this category (e.g., issues with installing the library in different platforms $Q_{61883115}$). From the beginning of 2017 to mid-2018 AutoML cloud platform's bot development-related queries were quite popular (e.g., "Filling data in response cards with Amazon Lex" in $Q_{47970307}$). Around that time, Amazon Lex (2017) was released, a fully managed solution to design, build, test and deploy conversational AI solutions. From our analysis, we can see the rise of Pipeline Automation-related discussion from the beginning of 2019 (e.g., issues related to the deployment pipeline in $Q_{55353889}$). We also notice a significant rise in the number of questions related to model load and deployment from 2019 to 2022 (e.g., "Custom Container deployment in vertex ai" in $Q_{69316032}$).

Model This is the second largest AutoML topic category, with around 28% of AutoML-related questions and four AutoML topics. From Figure 4.14, we can see that, similar to MLOps; the Model topic category began to gain popularity from the start of 2017. The dominant topic in this category is the Model performance improvement-

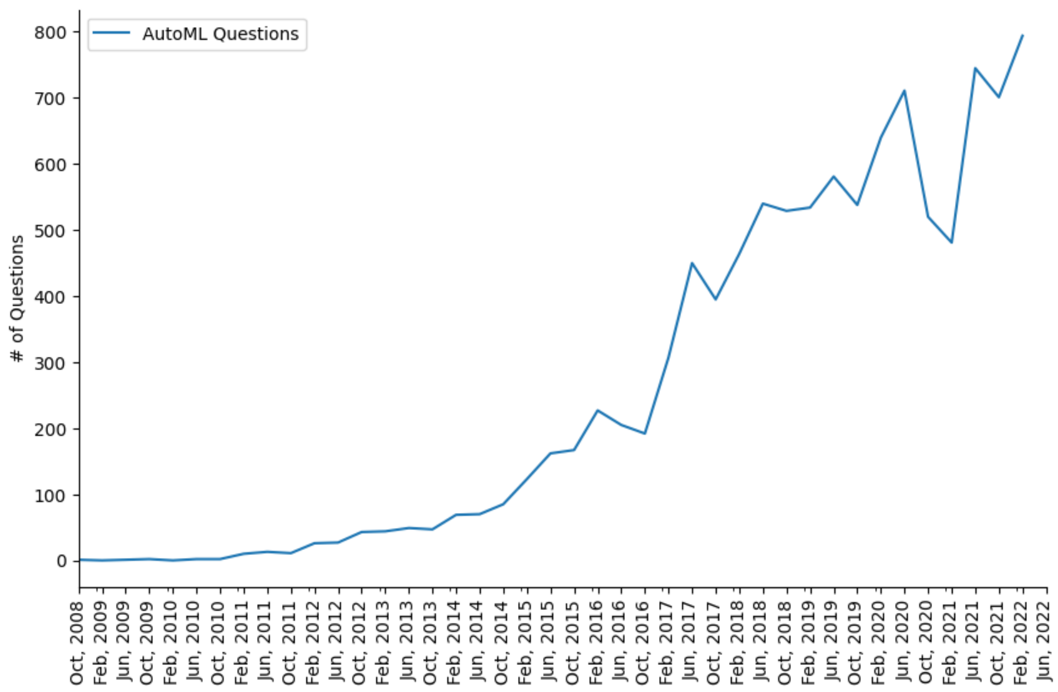


Figure (4.13) The evolution of overall AutoML related questions over time.

related queries using AutoML services (e.g., "Grid Search builds models with different parameters (h2o in R) yields identical performance - why?" in $Q_{47475848}$, "How to understand the metrics of H2OModelMetrics Object through h2o.performance" in $Q_{43699454}$). From our analysis, we see around a 200% increase in the number of questions in Model training and monitoring topic from 2021 to mid-2021. So practitioners ask queries regarding training issues (e.g., "Training Google-Cloud-Automl Model on multiple datasets" in $Q_{68764644}$, $Q_{56048974}$) from practitioners in recent times. Other topics related to model design, model implementation, and debugging evolved homogeneously around this time.

Data This is the third largest AutoML topic category, with around 27% of AutoML-related questions and three AutoML topics. The Data Management topic is the most dominant topic in this category. From our analysis, we can see around a 60% increase in the number of queries on cloud data management, especially after Splunk cloud becomes available on the google cloud platform. So, we can see developers' queries related to data transfer (e.g., in $Q_{63151824}$), REST API (e.g., in $Q_{63152602}$), API rate limitation (e.g., in $Q_{63292162}$) becomes popular. Other topics from this category evolve homogeneously over time.

Documentation This is the smallest AutoML topic category with around 2% of AutoML-related questions and one AutoML topic. We find that Since 2015 questions from this category shifted slightly from API-specific details (e.g., $Q_{35562896}$) to documentation-related limitations for cloud deployment (e.g., $Q_{59328925}$), containerization (e.g., $Q_{68888281}$).

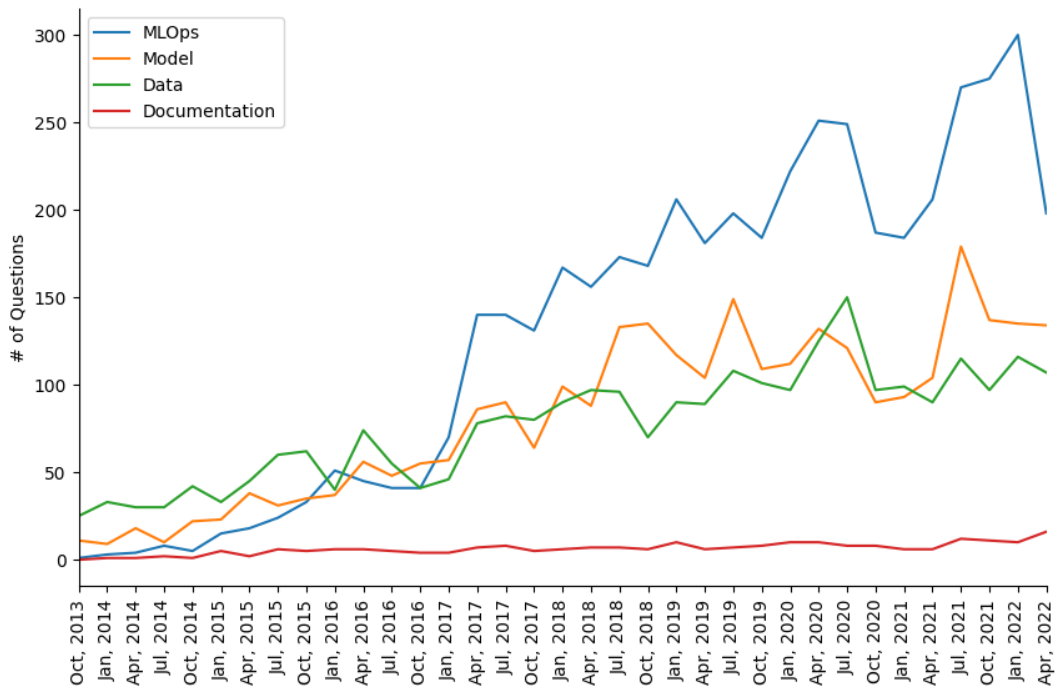


Figure (4.14) The evolution of SO questions on AutoML topic categories over time.

4.5.2 Top AutoML service providers

In Figure 4.15, we present the evolution of the top seven AutoML solutions in terms of the number of SO questions over the past decade. In our dataset, Amazon Sagemaker [35], is the largest AutoML platform containing around 20% of all the SO questions followed by Splunk (19%) [228], H2O AI (17%) [119], Azure Machine Learning [36], Amazon Lex [33], Google Cloud AutoML [114], and Rapid Miner [207]. Among these AutoML service providers, H2O AI AI began in mid-2017 with the release of their H2O.ai driverless AI cloud platform, but since then, it has been steadily losing practitioners' engagement in SO. We also observe that from its release in 2017 AWS SageMaker cloud platform remains dominating, with a notable surge occurring in the middle of 2021. Other AutoML platforms likewise demonstrate an overall upward trend.

AutoML service providers' support on SO discussion. Our analysis shows that popular AutoML providers actively follow SO discussions and provide solutions. For example, the H2O AutoML team officially provides support for practitioners' queries in SO (e.g., $A_{51527994}$), and thus they provide insights into our current limitations (e.g., $A_{52486395}$) of the framework and plans for fixing the bugs in future releases (e.g., Fig. 4.17 in $A_{51683851}$). Similarly, we also find the development team from Amazon Sagemaker also actively participating in the SO discussion (Fig. 4.18 in $A_{55553190}$). In Figure 4.16, we present a bar chart with the total number of questions vs. the percentage of questions with accepted answers for each of the top seven AutoML service providers. We can see that Amazon SageMaker has 2,053 questions, and only 33% of its questions have accepted answers. From Figure 4.16, we see H2O AI has

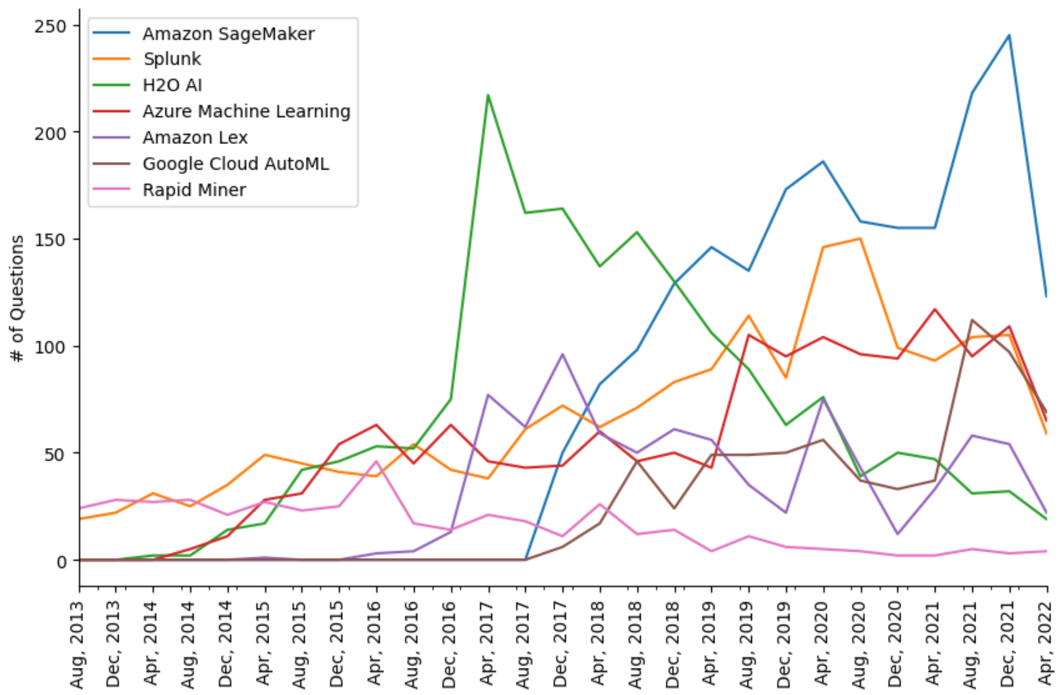


Figure (4.15) The evolution of questions for top AutoML services.

the highest percentage of questions with accepted answers (42%), followed by Rapid Miner (40%), Azure Machine Learning (38%), Splunk (36%), Amazon SageMaker (33%), Google Cloud AutoML (32%), Amazon Lex (31%).

4.5.3 Traditional ML vs AutoML challenges

With with popularity of ML and its widespread adoption, there are several studies on the challenges of machine learning development [14, 42], deep learning development [124], DL frameworks/libraries [134]. However, none of this research addresses the challenges of the low-code machine learning approach, i.e., AutoML. In this section, we compare the conclusions of these past studies on the border ML domain with our findings regarding the AutoML challenges. As AutoML is a subset of the ML domain, we find that many issues are comparable while others are unique. This finding is intriguing in and of itself, given that AutoML was intended to mitigate some of the shortcomings of traditional ML development and make ML adoption more accessible. Our analysis shows that AutoML tools are successful in some circumstances but not all. For example, among AutoML practitioners, there is substantially less discussion about Model creation and fine-tuning than among all ML practitioners. However, deployment, library, and system configuration dominate both traditional ML and AutoML development. Future research from the SE domain can further ameliorate many of these challenges by providing a guideline on better documentation, API design, and MLOps. We present a detailed comparison of some of the most significant ML development challenges faced by traditional ML versus AutoML practitioners.

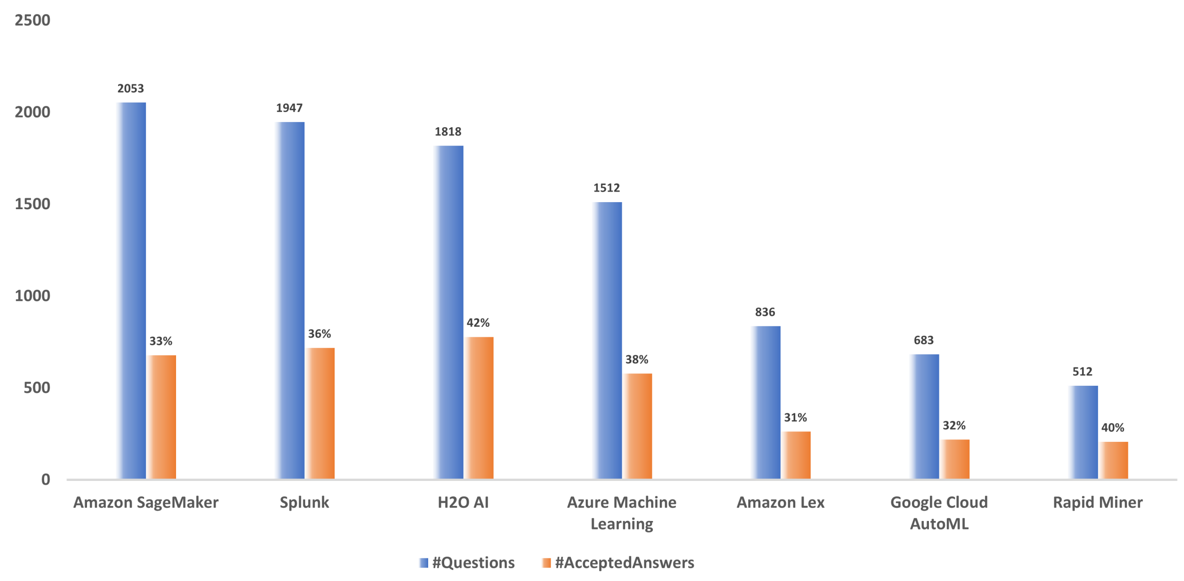
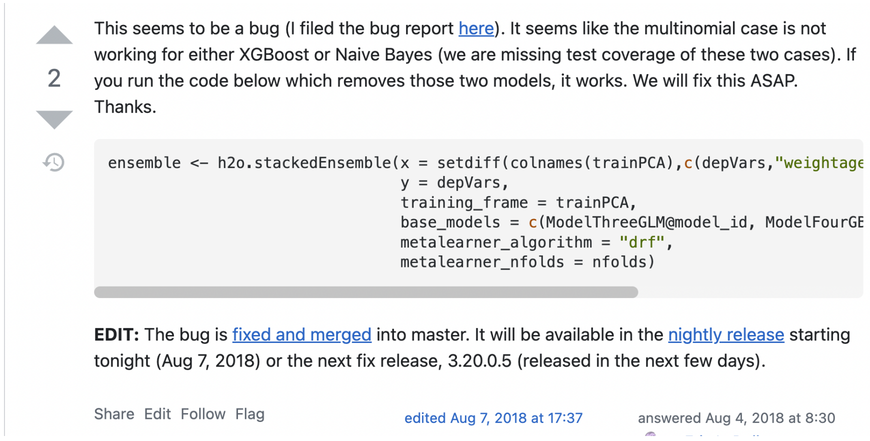


Figure (4.16) The status of top platforms and their percentage of accepted answers.


2

This seems to be a bug (I filed the bug report here). It seems like the multinomial case is not working for either XGBoost or Naive Bayes (we are missing test coverage of these two cases). If you run the code below which removes those two models, it works. We will fix this ASAP. Thanks.

```
ensemble <- h2o.stackedEnsemble(x = setdiff(colnames(trainPCA),c(depVars,"weightage
                                y = depVars,
                                training_frame = trainPCA,
                                base_models = c(ModelThreeGLM@model_id, ModelFourGE
                                metalearner_algorithm = "drf",
                                metalearner_nfolds = nfolds)
```

EDIT: The bug is fixed and merged into master. It will be available in the nightly release starting tonight (Aug 7, 2018) or the next fix release, 3.20.0.5 (released in the next few days).

Share Edit Follow Flag edited Aug 7, 2018 at 17:37 answered Aug 4, 2018 at 8:30

Figure (4.17) An example of H2O's platform's support for resolving reported bugs ($A_{51683851}$).

Requirement Analysis. This contains a discussion on formulating an ML solution for a specific problem/task. These questions are related to understanding available data and the support for existing ML frameworks/platforms. An empirical study on the machine learning challenges in SO discussion by Alshangiti et al. [14] reports that ML developers have around 2.5% questions on these challenges in SO. Other studies [134, 134] also report similar challenges for ML developers. From our analysis (Section 4.4.2.3), we also find that AutoML practitioners have around 5.7% queries regarding the supported services from AutoML tools/platforms to meet their business requirements. Our analysis (Fig. 4.11) indicates that AutoML practitioners find these to be particularly hard (e.g., "Does AzureML RL support PyTorch?" in $Q_{64025308}$ or "Is there support for h2o4gpu for deeplearning in python?" in $Q_{67727907}$); therefore, AutoML providers should give more support in terms of available resources.

Data Processing & Management. This challenge relates to data loading, cleaning, splitting, formatting, la-

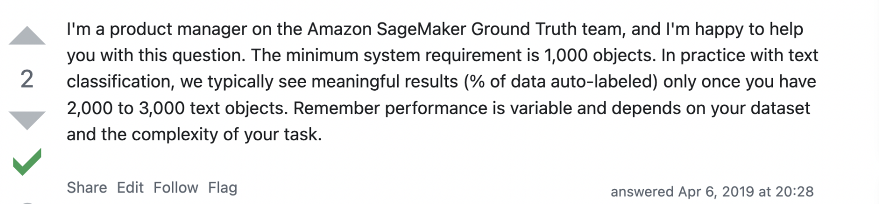

I'm a product manager on the Amazon SageMaker Ground Truth team, and I'm happy to help you with this question. The minimum system requirement is 1,000 objects. In practice with text classification, we typically see meaningful results (% of data auto-labeled) only once you have 2,000 to 3,000 text objects. Remember performance is variable and depends on your dataset and the complexity of your task.

Figure (4.18) An example of Amazon Sagemaker team monitoring and supporting SO practitioners' queries ($A_{55553190}$).

beling, handling missing values or imbalanced classes, etc. Previous studies report that ML developers find data pre-processing the second most challenging topic [14] and even some popular ML libraries lack data cleaning features [134]. Our analysis (Section 4.4.1.3 and Section 4.4.3.3, we find that this is quite a dominant topic for AutoML practitioners (i.e., 27% questions), but AutoML practitioners find these topics (Fig. 4.7) and MLLC phases (Fig. 4.11) less challenging. This finding suggests that AutoML service provide better abstraction and APIs for this data pre-processing and management pipeline.

Model Design. This contains some conceptual and implementation-related discussions on ML. This contains a discussion about ML algorithms/models, their internal implementation, parameters, etc. (e.g., Architecture of FCC, CNN, ResNet, etc., learning rate). Similar studies on ML challenges [134] report model creation-related questions as challenging. Questions regarding different ML algorithms are dominant in SO [42, 124], and Neural network architecture is a popular topic among practitioners and gaining popularity [42]. From our analysis, we find that this challenge is prevalent (4.2% questions) in AutoML practitioners (Fig. 4.5) and moderately challenging (Table 4.3).

Training & Debugging. This contains a discussion related to the actual training of the model on a specific train dataset. This contains a discussion about use cases or troubleshooting for a specific library via documentation or prior experience. Similar studies [42, 124, 134] report that coding error/exception topics – type mismatch, shape mismatch – are most dominant in SO, and resolving these issues requires more implementation knowledge than conceptual knowledge [14]. Our analysis suggests that AutoML practitioners also find model training and debugging-related questions (i.e., around 10%) most popular (Fig. 4.12). This suggests that both ML and AutoML practitioners require better support for bug detection [262].

Model Evaluation & Performance Improvement. This contains challenges related to evaluating the trained model via different metrics and improving the model's performance via different techniques and multiple iterations. Similar studies [134] also report that model-turning and prediction-related queries are less frequent in SO, even though significant ML research focuses on this step. More research is needed on practitioners' challenges. We find that AutoML practitioners have many queries (i.e., around 10%) regarding improving the performance of the model with fewer resources (e.g., "H2O cluster uneven distribution of performance usage" in $Q_{46056853}$, $Q_{62408707}$). AutoML aims to abstract model architecture-related complexities, but better explainability of the model's prediction may mitigate many of AutoML practitioners' concerns (e.g., "RapidMiner: Explaining decision tree parameters" in $Q_{23362687}$).

Model deployment. This contains challenges while deploying the model, scaling up as necessary, and updating the model with a new dataset. Chen et al. [65] report that ML developers face several challenges from Model export, environment configuration, model deployment via cloud APIs, and updating model. Guo et al. [118] evaluated prediction accuracy and performance changes when DL models trained on PC platforms were deployed to mobile devices and browsers and found compatibility and dependability difficulties. Similar studies [14, 134] on developers' discussion on SO on DL frameworks reported that developers find model deployment challenging [124]. From our analysis, we also find that (Fig. 4.12) AutoML practitioners find Model Load & Deployment (around 13% questions) to be one of the most difficult topics (e.g., "Azure ML Workbench Kubernetes Deployment Failed" in $Q_{46963846}$). This is particularly problematic as AutoML is expected to make model deployment and stability-related problems. Future research efforts from SE can help to mitigate these challenges.

Documentation. This challenge is related to incorrect and inadequate documentation, which is also quite prevalent in other software engineering tasks [4,4]. Similar to empirical studies [124,134] on the overall ML domain, we find that API misuse related to inadequate documentation or ML experts [14] is prevalent in all MLLC phases (e.g., In adequate documentation "API call for deploying and undeploying Google AutoML natural language model - documentation error?" in $Q_{67470364}$). This implies that researchers ML/AutoML should collaborate with the SE domain to improve these prevalent limitations [248].

4.5.4 Traditional vs. AutoML LCSD practitioners challenges

In Section 4.4.1, we present 13 AutoML-related topics organized into four categories based on an analysis of practitioners' discussions in SO. Similarly, in the previous Chapter 3, we present 40 traditional LCSD-related topics organized into five categories. In this Section, we provide a detailed comparison of the challenges of traditional LCSD practitioners vs AutoML practitioners. Table 4.6 provides an overview of 18 differences between traditional and ML LCSD challenges divided into six categories.

- **Customization.** We find that traditional LCSD practitioners struggle largely with UI and application logic customization programming issues. On the other hand, practitioners of low-code ML are less concerned about this.
- **Data.** Practitioners of traditional and ML low-code struggle with this. Traditional practitioner challenges concentrate mostly on database design and querying. In contrast, the key issues faced by ML low-code practitioners are preprocessing and exploratory data analysis of structured and semi-structured files.
- **Development & Debugging.** Traditional low-code practitioners suffer mostly from third-party RESTFul API

Table (4.6) The comparison between the struggles of traditional LCSD practitioners and AutoML practitioners.

Category	Topic	Traditional LCSD Practitioners' struggle	Machine Learning LCSD Practitioners' struggle
Customization	UI	We observe numerous practitioners discussing difficulties associated with dynamic form layout, dynamic page layout, dialogue box design, and so forth.	ML applications' primary focus is the ML model's performance; hence, we notice little to no discussion regarding UI design.
	Logic	We observe practitioners' discussion on dynamic event handling, data binding, and programming-related issues like pattern matching and date/time formatting, among others.	Typically, the trained model is integrated into a larger software application. However, we observe relatively little discussion addressing business logic implementation in AutoML tools.
Data	DBMS	We observe extensive debate regarding Database architecture, migration, replication, SQL CRUD operations, DB structure, and Entity relationship management.	Some cloud-based off-the-shelf data analytics systems utilize existing SQL or NoSQL data sources and occasionally process Petabyte-scale data.
	File	This topic includes the storage of structured and semi-structured files (CSV, JSON), as well as images files	The majority of ML data, is stored in CSV, JSON, or XML files, image data, etc. Thus, we uncover a substantial number of discussions relevant to this.
	Data Analysis	Traditional LCSD platforms contain minimal to no discussion of data analytics.	Many EDA approaches must be utilized, as well as data processing and exploration on the dataset to train ML models.
Development & Debugging	API Integration	Practitioners frequently discuss integrating external APIs, retrieving and processing API replies, integrating RESTful APIs, etc.	We do not observe AutoML practitioners discussing such concerns.
	Plug-in	Traditional LCSD platforms allow numerous third-party plugins for eSignature, text, and email processing, and practitioners have many questions regarding their compatibility and upgradeability.	We do not observe AutoML practitioners discussing such concerns.
	Debugging	Traditional LCSD practitioners discuss debugging-related application logic, data query, and user interface design challenges, among others.	AutoML LCSD practitioners discuss data exploration, model creation, and training-related debugging issues.
	Visualization	We find that practitioners have difficulty using interactive reports, graphs, and charts.	We observe a large number of inquiries concerning data visualization, visualization of the model's prediction, learning rate, etc.
Deployment & Maintenance	Deployment	Most practitioners struggle with where and how the developed application will be deployed — proprietary or third-party cloud, hosting settings, and SEO configuration.	How the trained model can be exported, deployed, or integrated with other applications is a major concern.
	Resource Management	The primary topic of discussion is how many requests can it process in parallel.	How much computing power it is taking (i.e., GPU/CPU utilization) is the main topic of discussion.
	Performance	Performance-related questions discuss how quickly the program operates, how many concurrent requests it can handle, and how to scale.	Concerning how well the trained model is working, how to increase performance and the best evaluation criteria.
Platform Adoption	Architecture	Traditional LCSD practitioners inquire about user role management, platform feature-related inquiries, Authentication & Authorization, and Authentication & Authorization.	Similarly, AutoML practitioners have questions regarding user management and model design for cloud-based AutoML solutions.
	Configuration	We observe practitioners' discussion on library installation, dependency management, Server configuration, SEO setting, etc.	Similarly, we find numerous questions regarding the installation and configuration of an appropriate development environment for AutoML tools. Many non-cloud-based AutoML practitioners struggle with this.
	CI/CD	Practitioners have difficulty with continuous testing, deployment, and batch job monitoring	We discovered a few queries related to regularly updating the model as additional data becomes available so that the model's performance does not deteriorate.
	Automated workflow	We observe discussions about automating certain components of the development process, such as UI design, testing, data storage maintenance, and scalability.	Practitioners of AutoML highlight their concerns with the transparency of end-to-end ML workflow automation.
Documentation	Inadequacy	We observe practitioners struggle with inadequate, incomplete, or incorrect documentation	Similarly, we observe practitioners struggle with inadequate documentation

integration and debugging programming. On the other hand, low-code ML practitioners struggle with coding-related challenges pertaining to data visualization and model debugging.

- **Deployment & Maintenance.** Our data indicates that this is the most difficult for traditional and machine-learning low-code practitioners. Traditional practitioners, on the other hand, focus mostly on cloud deployment and scalability-related issues. In contrast, low-code ML practitioners struggle with resource consumption and model performance-related issues that typically deteriorate with time if they are not fine-tuned with new data.
- **Platform Adoption.** Traditional low-code developers and AutoML developers deal with the limits of the design, configuration, and automated workflows of cloud-based platforms. If the platform does not support the required functionalities, it becomes extremely difficult to implement them.
- **Documentation.** In both LCSD services, practitioners have difficulty locating documentation that could have lessened their difficulties with the customization, deployment, maintenance, and platform adoption.

4.5.5 The application of the taxonomy of challenges

Now we discuss how the findings of this study, i.e., the taxonomy of discussed topics by the AutoML practitioners can be used to choose a new AutoML solution. First of all, the findings of this study help practitioners to better understand the tradeoffs of cloud vs non-cloud AutoML Solutions (4.4.4). The findings of this study can also help practitioners and project managers to get significant insights regarding choosing a particular low-code platform. In Section 4.5.2, we discuss top AutoML solutions and their community support on SO 4.5.2. From our analysis, new practitioners can have a better understanding of the discussed challenges (4.4.1), their distribution across SDLC phases (4.4.2), and their popularity and difficulty (4.4.3). These analyses can help AutoML practitioners to have a better understanding of the potential limitations of supported features across different ML workflows. For example, We can see H2O AI has accepted answers for 42% questions whereas Google Cloud AutoML has only 32%. We can see MLOps is the most difficult topic category which is dominated (i.e., 80%) by cloud-based AutoML solutions. Popular AutoML platforms also have greater community support, which is crucial for long-term maintainability.

4.6 Implications

In this Section, we summarize the findings that can guide the following three stakeholders: (1) AutoML Researchers & Educators to have a deeper understanding of the practical challenges/limitations of current AutoML tools and prioritize their future research effort, (2) AutoML platform/tool Providers to improve their services, learning resources, improve support for deployment, and maintenance, (3) AutoML Practitioners/Developers to gain useful insights of current advantages and limitations to have a better understanding of the trade-offs between AutoML tools vs Tradition ML tools, We discuss the implications in detail below.

AutoML Researchers & Educators. We find that the AutoML practitioners' challenges are slightly different from traditional ML developers (Section 4.5.3). Researchers can study how to improve the performance of AutoML core services, i.e., finding the best ML algorithm configuration or Neural architecture search. AutoML tools/platforms aim to provide a reasonably good ML model at the cost of huge computational power (i.e., sometimes 100x) as it needs to explore a huge search space for potential solutions [29, 274]. Some recent search approaches are showing promising results by reducing this cost by around ten fold [80, 130, 170, 198]. From our analysis (Section 4.4.3), we can see model performance improvement is a popular and difficult topic for AutoML practitioners (Table 4.2). Future research on the explainability of the trained model can also provide more confidence to the practitioners [127, 273]. Researchers from SE can also help improve documentation, API design, and design intelligent tools to help AutoML practitioners correctly tag SO questions so they reach the right experts [40, 42].

AutoML Vendors. In this research, we present the AutoML practitioners discussed topics (Fig.4.5), popularity (Table 4.2), and difficulty (Table 4.3) of these topics in detail. From Figure 4.12, we can see that Model load & deployment-related topic is one of the most popular and difficult topics among the practitioners. We can also see that Model training & monitoring-related questions are the most challenging topic and model performance, library/platform management, and data transformation-related topics are the most popular. AutoML vendors can prioritize their efforts based on the popularity and difficulty of these topics (Fig. 4.7). For example, expect for Model Load & Deployment (46 hours) and Model Training & Monitoring (32 hours) topics questions from other 11 topics have a median average wait for an accepted answer is less than 20 hours. We also find that topics under the data category are quite dominant and our analysis of AutoML tools and similar studies on other ML libraries [14, 134] report the limitation of data processing APIs that require special attention.

From Figure 4.11, We see that questions asked during model design and training related MLLC phase are quite prevalent and popular among the AutoML community. We can also see that AutoML practitioners find the requirement analysis phase (e.g., "How can I get elasticsearch data in h2o?" in $Q_{48397934}$) and Model deployment & monitoring phase quite challenging (e.g., "How to deploy machine learning model saved as pickle file on AWS SageMaker" in $Q_{68214823}$). We can also see popular development teams from popular AutoML providers such as H2O and Microsoft (Fig. 4.17, Fig. 4.18 in Section 4.5.2) actively follow practitioners discussion on SO and take necessary actions or provide solutions. Other AutoML tools/platform providers should also provide support to answer their platform feature-specific queries. It shows that the smooth adoption of the AutoML tools/platforms depends on improved and effective documentation, effective community support, and tutorials.

AutoML Practitioners/Developers. The high demand and shortage of ML experts are making low-code, i.e., the AutoML approach for ML development attractive for organizations. We can also see the AutoML-related discussion in SO a steady upward trend (Fig. 4.13). AutoML tools/platforms provide great support for different ML aspects such as data pre-processing and management but still lack support for various tasks especially tasks related to model deployment and monitoring. Our analysis can provide project managers with a comprehensive overview of the current status of overall AutoML tools (Section 4.5.2 and Section 4.5.3). Our analysis provides invaluable insight into the current strengths/limitations to project managers to adopt the AutoML approach and better manage their resources. For example, even with the AutoML approach setting up and configuring development environment and library management (e.g., "Need to install python packages in Azure ML studio" in $Q_{52925424}$), deployment trained models via cloud endpoint and updating model is still challenging. Our analysis also shows that the average median wait time to get an accepted solution from SO is around 12 hours for data management and model implementation/debugging-related tasks as development teams from some of these platforms also provide active support.

4.7 Threats to Validity

Internal validity threats in our study relate to the authors' bias while analyzing the questions. We mitigate the bias in our manual labeling of topics and AutoML phases by following the annotation guideline. The first author participated in the labeling process, and in case of confusion, the first author consulted with the second author.

Construct Validity threats relate to the errors that may occur in our data collection process (e.g., identifying relevant AutoML tags). To mitigate this, first, we created our initial list of tags, as stated in Section 4.3, by analyzing the posts in SO related to the leading AutoML platforms. Then we expanded our tag list using state-of-art approach [2, 7, 10, 11, 37, 215, 254]. Another potential threat is the topic modeling technique, where we choose $K = 15$ as the optimal number of topics for our dataset B. This optimal number of topics directly impacts the output of LDA. We experimented with different values of K following related works [2, 10, 11, 37]. We used the coherence score and our manual observation to find K's optimal value that gives us the most relevant and generalized AutoML-related topics [2, 10, 11, 254].

External Validity threats relate to the generalizability of our research findings. This study is based on data from developers' discussions on SO. However, there are other forums that AutoML developers may use to discuss. We only considered questions and accepted answers in our topic modeling following other related works [2, 10, 11, 254]. The accepted and best answer (most scored) in SO may be different. The questioner approves the accepted answer, while all the viewers vote for the best answer. It is not easy to detect whether an answer is relevant to the question. Thus We chose the accepted answer in this study because we believe that the questioner is the best judge of whether the answer solves the problem. Even without the unaccepted answers, our dataset contains around 14.3K SO posts (12.5K questions + 3.7K accepted answers). Nevertheless, we believe using SO's data provides us with generalizability because SO is a widely used Q&A platform for developers from diverse backgrounds. However, we also believe this study can be complemented by including discussions from other forums and surveying and interviewing AutoML practitioners.

4.8 Related Work

4.8.1 Research on AutoML

There are ongoing efforts to increase the performance of AutoML tools through better NAS approach [80, 130, 170, 198], search for hyperparameters and loss methods [153, 155, 155], automating machine learning via keeping human in the loop [151]. Truong et al. [242] evaluated few popular AutoML tools on their abilities to automate ML pipeline. They report that at present different tools have different approaches for choosing the best model and optimizing hyperparameters. They report that the performance of these tools are reasonable against custom dataset but they still need improvement in terms of computational power, speed, flexibility etc. Karmaker et al. [140] provides an overview of AutoML end-to-end pipeline. They highlight which ML pipeline requires future effort for the adoption of AutoML approach. There are other case studies on the use of AutoML technologies for autonomous vehicles [156], industry application [156, 156] anticipate bank collapses, predicting bank failures [5] where the authors evaluate current strengths and limitations and make suggestions for future improvements. These studies focus on improving the AutoML paradigm from a theoretical perspective, and they do not address the challenges that AutoML practitioners have expressed in public forums such as SO.

4.8.2 Empirical Study on Challenges of ML Applications

There has been quite some research efforts [82, 91, 94, 192] mostly by interviewing ML teams or surveys on the challenges of integrating ML in software engineering, explanability on the ML models [88,90]. Bahrampour et al. [38] conducted a comparative case of study on five popular deep learning frameworks – Caffe, Neon, TensorFlow, Theano, and Porch – on three different aspects such as their usability, execution speed and hardware utilization.

There are also quite some efforts on finding the challenges of Machine learning application by analysing developers discussion on stack overflow.

There are a few studies analyzing developer discussion on Stack Overflow regarding popular ML libraries. Islam et al. [134] conduct a detailed examination of around 3.2K SO posts related to ten ML libraries and report the urgent need of SE research in this area. They report ML developers need support on finding errors earlier via static analyser, better support on debugging, better API design, better understanding of ML pipeline. Han et al. [124] an empirical study on developers' discussion three deep learning frameworks – Tensorflow, Pytorch, and Theano – on stack overflow and GitHub. They compare and report their challenges on these frameworks and provide useful insights to improve them.

Zhang et al. [280] analysed SO and GitHub projects on deep learning applications using TensorFlow and reported the characteristics and root causes of defects. Cummaudo et al. [70] use Stack Overflow to mine developer dissatisfaction with computer vision services, classifying their inquiries against two taxonomies (i.e., documentation-related and general questions). They report that developers have a limited understanding of such systems' underlying technology. Chen et al. [65] conducts a comprehensive study on the challenges of deploying DL software by mining around 3K SO posts and report that DL deployment is more challenging than other topics in SE such as big data analysis and concurrency. They report a taxonomy of 72 challenges faced by the developers. These studies do not focus on the challenges of overall machine learning domain but rather on popular ML libraries, ML deployment, or a specific cloud-based ML service. Our study focuses entirely on overall AutoML related discussion on SO; hence, our analysed data, i.e., SO posts, are different.

There are also several empirical research on the challenges of machine learning, particularly deep learning, in the developers' discussion on SO. Bangash et al. [42] analyzes around 28K developers' posts on SO and share their challenges. They report practitioners' lack of basic understanding on Machine learning and not enough community feedback. Humbatova et al. [131] manually analysed artefacts from GitHub commits and related SO discussion and report a report a variety of faults of while using DL frameworks and later validate their findings by surveying developers. Alshangiti et al. [14] conduct a study on ML-related questions on SO and report developers' challenges and report that ML related questions are more difficult than other domains and developers find data pre-processing, model deployment and environment setup related tasks most difficult. They also report although neural networks, and deep learning related frameworks are becoming popular, there is a shortage of experts in SO community. In contrast, our study focuses on AutoML tools/platforms rather than boarder machine learning domain in general. In this study, we aim to analyse the whole AutoML domains, i.e., AutoML-related discussion on SO rather than a few specific AutoML platforms. Our analyzed SO posts and SO users differ from theirs, and our findings provide insight focus on AutoML's challenges.

4.8.3 Research on Topic Modeling & SO discussion

Our reason for employing topic modeling to understand LCSD discussions is rooted in current software engineering study demonstrates that concepts derived from The textual content can be a reasonable approximation of the underlying data [64, 231, 232]. Topic modelling in SO dataset are used in a wide range of studies to understand software logging messages [154] and previously for diverse other tasks, such as concept and locating features [67, 202], linking traceability (e.g., bug) [24, 206], to understand the evolution of software and source code history [129, 235, 236], to facilitate categorizing software code search [237], to refactor software code base [46], and for explaining software defects [63], and various software maintenance [230, 231]. The SO posts are used in several studies on various aspects of software development using topic modeling, such as what developers are discussing in general [43] or about a particular aspect, e.g., low-code software developers challenges [10, 11], IoT developers discussion [254], docker development challenges [125], concurrency [7], big data [37], chatbot [2], machine learning challenges [14, 42, 65], challenges on deep learning libraries [124, 134].

4.9 Summary

AutoML is a novel low-code approach for developing ML applications with minimal coding by utilizing higher-level end-to-end APIs. It automates various tasks in the ML pipeline, such as data prepossessing, model selection, hyper-parameter tuning, etc. We present an empirical study that provides valuable insights into the types of discussions AutoML developers discuss in Stack Overflow (SO). We find 13 AutoML topics from our dataset of 14.3K extracted SO posts (question + acc. answers). We extracted these posts based on 41 SO tags belonging to the popular 18 AutoML services. We categorize them into four high-level categories, namely the MLOps category (5 topics, 43.2% questions) with the highest number of SO questions, followed by Model (4 topics, 27.6% questions), Data (3 topics, 27% questions), Documentation (1 topic, 2.2% questions). Despite extensive support for data management, model design & deployment, we find that still, these topics are dominant in AutoML practitioners' discussions across different MLLC phases. We find that many novice practitioners have platform feature-related queries without accepted answers. Our analysis suggests that better tutorial-based documentation can help mitigate most of these common issues. MLOps and Documentation topic categories predominate in cloud-based AutoML services. In contrast, the Model topic category and Model Evaluation phase are more predominant in non-cloud-based AutoML services. We hope these findings will help various AutoML stakeholders (e.g., AutoML/SE researchers, AutoML vendors, and practitioners) to take appropriate actions to mitigate these challenges. The research and developers' popularity on AutoML indicates that this technology is likely widely adopted by various businesses for consumer-facing applications or business operational insight from their dataset. AutoML researchers and service providers should address the prevailing developers' challenges for its fast adoption. Our future work will focus on (1) getting AutoML developers' feedback on our findings by interviews or surveys, and (2) developing tools to address the issues observed in the existing AutoML's data processing and model designing pipeline.

Chapter 5

AutoGeoML: A Low Code Machine Learning Toolkit for Geothermal Software

In the previous Chapter 4, we presented the challenges of low-code ML, i.e., AutoML practitioners, by analysing their discussion in SO. This Chapter aims to complement those findings by conducting a case study of applying popular ML and AutoML frameworks to develop ML models to predict downhole drill string vibration during horizontal drilling. We present insights about the strength and weaknesses of AutoML tools and the role of domain experts in ML workflow automation. Based on end users' and stakeholders' requirements, we developed AutoGeoML, an open-source low-code framework to predict downhole vibration. We offer some design implications for future AutoML tool development. We find that instead of focusing on an end-to-end API for the Automated ML pipeline, AutoML vendors can focus on a range of high-level APIs for each ML workflow for domain experts with different goals such as customizability, interoperability, and simplicity.

5.1 Introduction

Machine Learning (ML) shows great promise in solving various problem image classification to Automated driving to Natural Language Processing (NLP). But this is a cumbersome iterative process which requires a lot of expertise to develop a decent model and thus only accessible to a few. ML engineers have to experiment with many repetitive tasks for data pre-processing, model designing, feature engineering, hyper-parameter optimization and finally, empirically evaluating and deciding the best model for that particular problem or dataset.

There is a huge shortage of ML experts compared to the market demands. So, in recent times, there has been a rapid rise in popularity for AutoML, which is a new field in ML aiming to automate different ML pipelines. AutoML aims to make ML more accessible to domain experts and ML engineers. From the very has been a research approach to automate cumbersome hyperparameter optimization searches for traditional ML algorithms. Over time, it evolved to automate other ML pipelines, such as data pre-processing, feature engineering, model selection and training, and even model deployment and interpretability. The ultimate goal of AutoML is to offer off-the-shelf solutions so that domain experts can develop decent ML applications just from the dataset without an in-depth understanding of ML.

To understand the effectiveness of the automated ML workflow, we conduct a case study of developing an ML solution for predicting downhole vibration during vertical drilling. We collaboratively work with domain experts from the industry and academia. We systematically collect different requirements (e.g., scope, customizability, model training and deployment) from a software engineering perspective. We explore the features of different AutoML tools that satisfy the operational requirements of our end user. Then we develop a Vibration prediction model that provides the right amount of abstraction (i.e., automation), flexibility, and extendability. Our approach gives us useful insights into requirements and expectations from domain experts and ML engineers and the strength and limitations of existing AutoML tools. It also provides us with valuable insight regarding human involvement (e.g., domain experts and ML engineers), ML workflow automation and releasing an ML software application to end users.

We find that human ML engineers and domain experts are indispensable in developing machine learning software applications (MLSA). Human domain expertise and intuition help find current AutoML tools' limitations. Furthermore, human involvement is necessary for the interpretability and reliability of the developed model. Similar to other studies [110, 271], we find that instead of focusing on end-to-end ML models from the dataset, the AutoML providers may focus on the human-in-the-loop approach for automating different ML workflow. From our case study, we find that a complete end-to-end API from dataset to ML model is neither feasible nor necessary; instead, various high-level abstraction over each ML pipeline is more helpful.

Some other studies [98, 110, 110, 271] have also advocated for human-guided ML automation. In this study, we employ a bottom-up approach to collect requirements and iteratively develop ML solutions by continuously collaborating with domain experts and other software engineers who integrate this ML model into a large software application. This gives us unique insights into an Automated ML library's customizability, extendability and maintainability. To satisfy customer requirements for data pre-processing and model customizability, we have to develop AutoGeoML, which is a wrapper over existing ML (e.g., TensorFlow) and AutoML library (e.g., AutoSklearn). It provides abstract APIs for domain experts to provide feedback for each machine-learning pipeline. We also highlight the current limitations and recommendations for future improvement. We hope these insights and recommendations will guide future research efforts for AutoML development.

This chapter is organized as follows. Section 5.2 presents some works that are related to this study. Section 5.3 describes the scopes and requirements for the machine learning model. Section 5.4 reports the motivation, approach, and high-level design of our proposed AutoGeoML. Section 5.5 provides discussion, implications of our findings and threats to validity. Section 5.7 concludes this chapter.

Replication Package: The code and data are shared in `https://github.com/disa-lab/Thesis-Alamin-MSc/tree/main/Chapter_5`

5.2 Related Work

AutoML services focus on end-to-end automated ML workflow. In contrast, human-in-the-loop ML tools focus on incorporating humans into different aspects of ML workflow, such as data collection, model debugging, model evaluation, etc. So, related research in the following areas is relevant to this study: (1) AutoML Services, (2) Human in the loop ML development, and (3) Research on challenges MLSA application.

AutoML Services. There is quite a several research on open-source AutoML libraries/frameworks both from academia and industry, such as AutoSklearn [100] from the University of Freiburg, TPOT [184] from the University of Pennsylvania, H2O [150] from H2O company, AutoKeras [136] from Texas A&M University. Some of these AutoML tools are designed for traditional ML models (e.g., AutoSklearn), and some for deep learning models primarily focused on neural architecture search (e.g., AutoKeras, Ludwig). These tools provide great flexibility regarding customizability and integration of larger software systems via APIs. Usually, they lack MLOps support, i.e., setting up & managing the development environment, model deployment & monitoring. There are also some AutoML cloud provider solutions such as Google Cloud AutoML [114], Azure AutoML [36], Amazon Sagemaker [35] which provides off-the-shelf solutions for ML by specially targeting businesses by offering computational resources, model deployment, and interpretability support.

human in the loop ML development. There are many research paradigms for incorporating human experts into ML workflow automation, such as interactive machine learning [79, 98, 261], Human involvement in ML Automation by Xin et al. [271], human-guided machine learning [110], human in the loop perspective for AutoML [151]. It involves domain and human experts to evaluate and improve the model in an iterative approach [16, 101]. It enables

ML practitioners without in-depth ML expertise to provide feedback and improve the model's performance iteratively (e.g., improving feature engineering [98], model transparency & interpretability [264, 267], model debugging [186]) often via visual interface [16, 276]. In this study, we present the limitations of end-to-end automation for AutoML tools, and they can lower the entry barrier for ML development for ML and domain experts.

research on challenges MLSA application. Several types of research focus on the current challenges of machine learning in software engineering [85,86,96], empirical research on the best practices of integrating AI capabilities [83], developers survey on challenges faced during different SDLC [94]. There are also quite a few studies on the quality assurance of ML models like reliability, transparency, trustworthiness, etc. [87, 89, 90, 90, 92, 96]. There are few research on the MLSA quality assurance challenges [9,84,93,95,95]. For this project, we incorporate our ML solution into an MLSA application and highlight our challenges.

5.3 Background

5.3.1 Vibration Forecasting

There are three forms of drill string vibration: axial, lateral, and torsional. Over time, they cause the drill string and other equipment to deteriorate [182]. It indicates that the drilling machine is not performing efficiently, which leads to poor drilling performance and perhaps damage, which can be quite expensive from an operational standpoint. Adding a high-frequency sensor to obtain real-time vibration data is highly costly, and as a result, they are rarely used. This project aims to construct a machine learning (ML) model to forecast down-hole vibration using surface parameters. Related research [182] explored supervised ML techniques and deep learning models [278] for this purpose. This study aims to create a high-performing machine learning (ML) model and a high-level, low-code automated pipeline so that domain experts can easily train and construct a downhole vibration prediction model for new drilling rigs.

5.3.1.1 Dataset Overview

$$Vibration = \sqrt{axialrms^2 + tangentialrms^2 + lateralrms^2} \tag{5.1}$$

Our data comes from sensors from five drilling rigs. It has in total of 3.8 Million rows of data; however, after filtering, we found 73K rows of active drilling data. Our data is tabular and delivered in a CSV file format. Some of the known surface sensor attributes are: "Hole Depth", "Bit Depth", "Block Height", "Weight on Bit", "Differential Pressure", "Standpipe Pressure", "Total Pump Output", "Mud Flow Rate", "RPM", "Hook Load", "Torque". It also contains downhole attributes (e.g., "AxialRMS", "TangentialRMS", "LateralRMS", "RPMMeds") from downhole sensors, which is used to calculate our target variable, i.e., downhole *vibration* using the equation 5.1. Following related work [182], we mainly focus on three surface feature attributes: "ROP", "Torque", "Weight on Bit (WOB)",

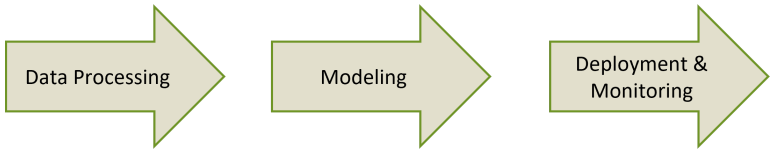


Figure (5.1) A high-level overview of automation requirements for different ML pipelines.

and the downhole target variable *vibration*.

5.3.2 Requirement collection & Scope definition

Requirement analysis is a method for gathering the final product's needs and expectations from the end user's perspective. We took the essential measures from the start to define the product's requirements and objectives. For almost six months, we communicated weekly, either via zoom or in person, with stakeholders, domain experts on drilling, and possible end users to define the scope of this research project. From a high-level perspective, the overall objective of this project is depicted in three steps (i.e., Figure 5.1): (1) *Data Collection & pre-processing*, i.e., tasks related to data collection, cleaning, labelling, transformation, (2) *ML Model design & training*, i.e., tasks related to training ML models on best configuration on pre-processed data, (3) *Using the model's prediction in the decision-making*, i.e., model evaluation, interpretation, deployment, monitoring, and updating with new data.

Table (5.1) The summary of the requirements of the system

ID	Type	Requirement	Description
R1	Non-functional	Exploratory Data Analysis (EDA)	It refers to data analysis, detecting anomalies, and verifying assumptions.
R2	Non-functional	Automated pipeline	It refers to a high-level low-code pipeline to easily develop vibration prediction models on different datasets.
R3	Non-functional	Modular design	refers to the flexibility to update each module independently and incorporate the solution in a larger system.
R4	Non-functional	Model interpretability	It refers to understanding what the model has learned and its justification for the prediction. Model Understandability & and visibility is an essential pre-condition.
R5	Non-functional	Computational cost	The trained model needs to be updated in real-time with new data, and thus the training time should be computationally within acceptable limit
R6	Functional	Data filtering	refers to the initial data exploration to filter out anomalous and missing data.
R7	Functional	Data pre-processing (i.e., windowing)	It refers to splitting the dataset into different windows, i.e., bins so that the correlation between features and the target variable holds.
R8	Functional	Model Design & Training	It refer to the ability to train shallow or deep learning ML models
R9	Functional	Model fine-tuning	It refers to updating the model with only new or in combination with old data drilling data, preferably in real-time.

System requirement. System requirements describe the features and characteristics of the product to meet the needs of the users (i.e., stakeholders) and business. They largely fall under two categories: (1) *Functional requirement*, i.e., what the final product must do, i.e., what features and functionalities it should have, (2) *Non-functional requirements*, i.e., the general properties/characteristics the system should have.

Table 5.1 shows the summary of the nine high-level requirements gathered from discussions with domain experts and other stakeholders through weekly meetings over four months. Among the requirements, the first 5, i.e., R1-R5, are non-functional requirements, and the last 4, e.g., R6-R9, are functional requirements. We describe our findings for these nine requirements under two categories below.

• **Non-functional requirements** refers to different system attributes such as performance, maintainability, security, scalability, customizability, etc. It refers to the constraints on how the system should meet its business goals. After elaborate discussion, we found five non-functional requirements for this project. They are: (1) R1. Exploratory Data Analysis (EDA), (2) R2. Automated pipeline, (3) R3. Modular design, (4) R4. Model confidence/interpretability, (5) R5. Computational cost. We describe each of these non-functional requirements in brief below.

- **R1. Exploratory Data Analysis (EDA)** refers to the critical systematic process of data formatting, transformation, data cleaning, time series test set generation, and removing abnormalities. It helps data scientists investigate the data's pattern, detect anomalies, and present summary statistics and graphical representations. Our goal was to understand the dataset better and verify the assumptions, i.e., the relationship between the feature and target variable. For example, from the physical property, we knew some features are positively (e.g., WOB) and negatively (e.g., Torque) co-related with the target variable. To that extent, we employed statistical analysis, Pearson correlation [68], Spearman correlation [53] and presented it with relevant stakeholders. After this, our goal was to develop an ML model based on these insights.

- **R2. Automated pipeline.** In software development, the automated pipeline process helps to complete similar tasks quickly. They streamline the operation by automating repetitive, time-consuming tasks. In this case, the requirement was to systematically analyze and develop a process that will allow the domain experts to train and update the downhole vibration prediction model as new data from new drilling rigs become available.

- **R3. Modular design** refer to the idea that divides a larger system into smaller subsystems, i.e., modules, so that they can be independently adjusted, changed, and merged with other modules or systems. In software engineering, modular architecture increases productivity, costs, and scalability. For this study project, it was necessary to create Vibration prediction ML modules with flexible data input and model output modules. The ultimate objective was to incorporate the ML module into a broader web application; hence, the data intake pipeline must be decoupled, as data can originate from many sources and in various formats. Likewise, the model's prediction can be represented and stored in various ways within the web application.

- **R4. Model interpretability/confidence** ML models are designed and trained to optimize an objective/cost function. In our case, this cost function (e.g., Root Mean Squared Error (RMSE)) can not fully capture the real-world cost of the model's prediction and decisions. Model interoperability allows us to understand better what the model has learned and its justification for the prediction. To apply the model's prediction, the stakeholders need to trust it. For this project, the requirement was to explain how the model makes its prediction and how confident it is. Understandability & visibility. Our target model is quite critical if they are deployed professionally. For example, there will be substantial financial loss in case of a wrong prediction.

One of the requirements is to save the trained model in a particular location so that the model's prediction may be explained using various tools, such as LIME [78]. LIME, which stands for local interpretable model agnostic explanations, is a black box ML model tool that facilitates the comprehension of each prediction. The requirement is to report how much weight the trained model gives to the feature variables so that domain experts may validate their knowledge and have greater confidence in the model's predictions.

- **R5. Computational cost.** The computational cost for machine learning is incurred during training, whereas the cost of making a prediction is fairly minimal. As we dig deeper into horizontal drilling, the underlying process changes, necessitating a model update as soon as new data becomes available. For this project, it is necessary to be able to update the model often and quickly (e.g., within 5-10 minutes) so that we can always get the most accurate prediction with the model updated with the most recent data.

• **Functional requirements.** It defines the function of a system. This may contain technical details, data management, computation, and other tasks that enable the system to achieve its target goal. After collaborating with domain experts and other stakeholders, we find various functional requirements for this project. After elaborate discussion, we found four functional requirements for this project. They are: (1) R6. Data filtering., (2) R7. Data pre-processing (i.e., windowing), (3) R8. Model Design & Training, (4) R9. Model fine-tuning. These four functional requirements are grouped into three categories: (1) Data, (2) Model, and (3) MLOps. We describe each of these four non-functional requirements in brief below.

Data. This contains the requirements related to data collection, filtering, and pre-processing. There are quite a few functional requirements from this category, but two of them are the most prevalent: (1) R6. Data filtering, and (2) R7. Data pre-processing.

- **R6. Data filtering.** We received our data in the form of multiple CSV files with all the attributes described in Section 5.3.1.1. It contains many rows of data without any drilling occurs. We were required to apply different filtering techniques such as (1) Rolling window-based filtering, (2) Percentile-based filtering, (3) Standard deviation-based filtering to filter out anomalous and missing data. The high-level goal is to design a semi-automated approach to incorporate human experts to ensure the dataset's quality and weed out the anomalous portion of data.

- **R7. Data pre-processing (i.e., windowing).** One of the characteristics of the dataset is that the correlation between different features and the target variable changes over time in the dataset as we dig deeper. From the physical aspect of drilling, soil and rock property change with deeper drilling. So, instead of using the whole dataset uniformly, we divided the whole dataset into different bins: (1) time-based binning, or (2) depth-based binning. These small bins comprise only a few minutes or meters of drilling data for which the correlation between features and the target variable holds. Therefore, partitioning the entire dataset into bins, or windows, is a prerequisite for training machine learning models.

Model. This contains user requirements related to flexibility and customizability of model selection and feature engineering.

- **R8. Model Design & Training.** The requirement was to provide a high-level abstraction over the complexities of choosing the right ML algorithms, feature engineering, hyperparameters configuration, and Model training, i.e., applying the AutoML tools. One simple requirement was this VibrationML Model should be able to with any number of training data rows, and it should accept a variable number of features (i.e., in some cases training the Model with subset feature variables). To that extent, we experimented with several approaches: (1) classical ML algorithms such as Linear Regression, Random Forrest, Support Vector Machine (SVM), (2) as our dataset is sequential (time-series) we also experimented with RNN-based deep learning model, (3) finally we experimented with AutoML tools (e.g., Auto-SkLearn and AutoKeras).

MLOps. It contains the requirement related to the operational aspects of ML, i.e., saving the Model, fine-tuning it against new data, and exporting & deploying the trained Model for real-time prediction. There are quite a few requirements from this category, but one of them is the most relevant, i.e., Model fine-tuning.

- **R9. Model fine-tuning.** The requirement was that initially, the Model would be trained on previous drilling data and be used to predict new drilling. But as soon as new data becomes available, the Model needs to be updated, i.e., fine-tuned for better prediction.

5.4 AutoGeoML

This Section describes a high-level overview of our low-code human-in-the-loop AutoGeoML library. A human-in-the-loop machine learning tool refers to a paradigm where the domain experts interactively define and develop a machine learning model through iterative cycles of reviews and development (Fig. 5.2). Designing an effective low-code interface requires close collaboration between ML and domain experts. The design of AutoGeoML is inspired by the requirements and constraints of end users as described in Section 5.3.1. However, these design principles can be applied to similar systems with similar requirements.

5.4.1 Design Motivation

In this Section, we discuss our approach to developing a low-code vibration prediction model addressing the requirements and scope described in Section 5.3.2. Now we discuss our studied ML frameworks, their strengths and weaknesses to meet our goals. The first design decision regarding our ML model was whether we should opt for a cloud vs. non-cloud (i.e., library/framework) solution. Based on the user requirement, we had to make a few design choices, but two of them are the biggest ones:

- *Design choice 1: Tools vs. cloud solutions.* To train our model on the dataset, we must first decide whether to use API-based AutoML frameworks or cloud-based AutoML platforms.
- *Design choice 2: AutoML solutions vs. hand-designed models.* Should we use AutoML solutions (i.e., black box feature and model selection and hyper-parameters optimization) or a simplified hand-designed ML model to suit user requirements?

5.4.1.1 Cloud vs. non-cloud solutions

AutoML cloud provides features that satisfy many functional and non-functional requirements, such as exploratory data analysis, end-to-end automated pipeline, and model interpretability. Most hosted AutoML services offer model interpretability by providing feature importance, holistic visualization of model architecture, and a decision-making process for a particular model. AutoML cloud platforms provide the necessary support to automatically generate various graphs and charts, summary statistics, leaderboards, etc. However, in this case, we needed to choose an API-based ML framework rather than a cloud-based end-to-end solution because, after model training, we were required to integrate a complex system and incorporate complex logic into the models' output.

5.4.1.2 AutoML frameworks vs. hand-designed models

Eventually, the domain technical experts preferred to go for hand designed ML model rather than AutoML tools because both of them provided a similar level of performance but using a lower-level API of SKlearn provided us with much better customizability and transparency. Feature Engineering: Feature selection and engineering are the most automated pre-processing task. It involves different techniques such as developing complex new features by combining existing new features, one hot encoding []. We find AutoML tools support quite insufficient in this regard. For this project, domain experts from our academia and industry suggested several derived features that helped achieve desirable results. These derived features are implemented using physics-supported mathematical equations. Thus, their significance and effects are known to the domain experts; thus, they were the preferred choice compared to AutoML tools' obscure generation of intermediary features.

• **Studied ML Frameworks** Our experimental model will be merged into a larger web application. Hence we needed an API-based machine learning library or framework for this task. Our objective is to design only the ML module, which will be visualized by a third-party module when it is incorporated into a web application. For this research, we explored the following four ML and AutoML libraries to satisfy our user requirements:

- SKlearn [225]: It is an open-source python-based ML library for various ML tasks such as classification, regression, and clustering. It features various algorithms such as SVM, Random Forrest, k-means, and gradient boosting. It works very well with popular python-based numerical scientific NumPy [183].
- AutoSKlearn [99]: is an AutoML toolkit and replacement for the Scikit-learn estimator. It frees ML practitioners to form algorithm selection and hyperparameter tuning by leveraging Bayesian optimization, ensemble construction, and meta-learning.
- AutoKeras [136]: is an AutoML tool for deep learning models using Keras API. It is developed and maintained by DATA Lab at Texas A&M university. It provides an easy-to-use API for different ML tasks, such as classification and regression for image, text, and structured datasets. It aimed to provide high-level abstraction so that a practitioner can develop a high-performing ML model only by specifying the location of the dataset and computational power (i.e., how many models it should explore)
- TensorFlow [223]: is an open-source ML and artificial intelligence library written in python. It especially focuses on training and interfacing for DNN networks.

Some of these libraries have better support for specific ML workflow. For example, SKlearn has better support for data pre-processing, especially for structured data. In comparison, other libraries provide better support for a complex neural architecture-based model for image, NLP, or time-series data.

• **AutoML solution's limitations.** AutoML allows novice practitioners to get business insight by applying ML techniques to the business dataset. However, in our case, this worked as a double-edged sword. It worked as a black box and provided a decent ML solution, however, at the cost of our model interpretability and configurability. However, its performance validated simple, easy-to-interpret hand-designed ML solutions (e.g., feature engineering and feature importance). After discussing the limitations and strengths of the studied AutoML frameworks/libraries and our solution approach for high-level models for vibration prediction to the domain experts and other stakeholders, we decided on the hand-designed model for the following seven reasons: (1) Involvement of Domain expertise, (2) Performance & computational cost, (3) Data exploration & filtering, (4) Feature Engineering & Model interpretability, (5) Customizability & maintainability, (6) Documentation & community support. we provide a brief description of each of these limitations below.

- **Involvement of Domain expertise.** AutoML solutions are developed to provide end-to-end automation of the data processing, model design, and model deployment pipelines. However, based on our user requirements, we've discovered that domain experts' participation in each pipeline phase gives significant insights. During exploratory data analysis (R1), data pre-processing (R7), Feature engineering (R4, R8), and model interpretability (R4) in the ML pipeline, for instance, domain experts are required to provide direct input. Therefore, we conclude that an end-to-end automated pipeline is neither required nor feasible. Varying ML stage end-users require different levels of abstraction and customizability. These requirements rely on the end-users and the particular domain or tasks. Therefore, after consulting with the relevant stakeholders for this project, we choose the appropriate level of abstraction for each stage of the machine learning (ML) pipeline where domain experts can utilize their existing knowledge.

- **Performance & computational cost.** We design and experiment with our hand-designed four ML models: (1) Linear Regression, (2) Random Forrest, (3) Support Vector Machine (SVM), and (4) time series LSTM model for a regression problem to predict the downhole vibration. We experiment with approximately 20 train-test sets and discover that no single model consistently performs well in all situations. Random Forrest and the LSTM model perform the best in automatic metric RMSE and visual observation. However, after discussion, we consider the random Forrest model for future experiments as it requires approximately ten times less computational power. However, after discussion, we consider the random Forrest model for future experiments as it requires approximately ten times less computational power.

  Then, we evaluate the performance of the AutoML tools (e.g., AutoSKlearn and AutoKeras). It enables rapid prototyping and experimentation with more machine-learning methods. Additionally, it enables us to explore a bigger search space with a distinct set of features. However, we find that their performance is comparable to that of our random forest model while being three to ten times more computationally expensive. It verifies that our manually crafted model and chosen features are operating optimally.

- **Data exploration & filtering.** Based on the user requirements (R1 and R6), we have determined that the AutoML tools' end-to-end pipeline over data pre-processing steps is unsuitable for our situation. They lacked the functionality to provide additional insight into data preparation steps, such as presenting statistics and various features' distribution (mean, standard deviation). Traditional Data processing libraries, such as Pandas, assisted us in meeting customer needs more effectively.

- **Feature Engineering & Model interpretability.** Feature selection and engineering are one of the most automated pre-processing tasks. It involves different techniques, such as developing complex new features by combining existing new features with one hot encoding. We find AutoML tools support quite insufficient in this

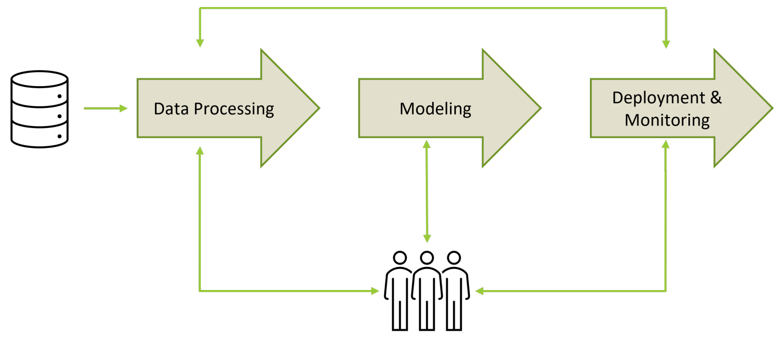


Figure (5.2) An high level overview of human-in-the-loop Machine learning [269].

regard. For this project, the domain experts from our academia and industry suggested several derived features that helped achieve desirable results. These derived features are implemented using physics-supported mathematical equations. Thus, their significance and effects are known (i.e., interpretable) to the domain experts; thus, they were the preferred choice compared to AutoML tools' obscure generation of intermediary features.

- **Customizability & maintainability.** Software maintainability refers to what extent the application can be repaired, extended, and enhanced. Lack of software maintainability, i.e., technical debt for machine learning applications, is quite high [91]. Since the project's use cases can change over time, one of the requirements is to have a module design (R3) and the capability to update the application over time. We observe that the developer community actively maintains and upgrades the traditional ML frameworks (i.e., TensorFlow, Sklearn) on GitHub. As a result, their stability and long-term maintainability (i.e., support) for future customization and upgrade are significantly superior to AutoML libraries.

- **Documentation & community support.** Compared to ML frameworks (e.g., TensorFlow or SKlearn), the documentation and tutorials for AutoML frameworks (e.g., AutoSklearn and AutoKeras) are rather scarce (e.g., missing out on details). In addition, we observe that the community support for debugging and customization-related discussions on Stack Overflow and GitHub is many times more for traditional ML frameworks than for AutoML libraries.

So, in summary, we do not need AutoML's end-to-end pipeline; rather, we need domain experts' input and customizability for every step of the ML pipelines. Thus to ensure the modular design and future maintainability, we design and develop AutoGeoML, which is developed using sklearn, TensorFlow, and Panda library and provides the appropriate amount of low-code abstraction and domain expert involvement to train and finetune downhole vibration detection model.

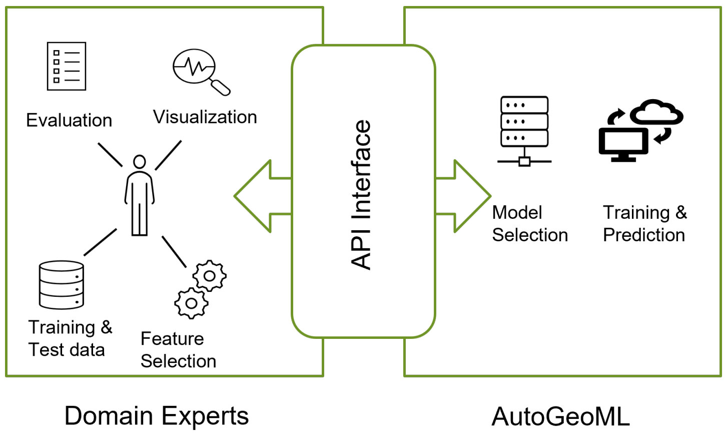


Figure (5.3)   A high level overview of human-in-the-loop AutoGeoML architecture.

5.4.2 Design Overview

The researchers are working to incorporate human expertise into different phases of ML pipeline [269] as depicted in Figure 5.2. Our AutoGeoML system has four key components: (1) Domain experts, (2) Data, (3) Model interface, and (4) Visual interface . These four components interact with each other and provide a semi-automated human-in-the-loop design framework. Figure 5.3 demonstrate the high-level architecture of VibrationML, where domain experts interact with our library via programming APIs. Domain experts utilize their insights on model training, model evaluation, and interpretation via the visual output. AutoGeoML provides modular programming APIs for visual representation and user interaction for effective model training. This domain-specific AutoML library design is similar to the generic human-in-the-loop interactive machine-learning design proposed by Dudley et al. [79]. We provide a brief description of each of these modules in brief below.

• **Domain experts.** In this system, domain experts are the main driving force. They have an in-depth understanding of the overall objectives of the system but lack a deep understanding of the ML techniques. The process necessary expertise to interpret the data and the model's prediction. Domain experts also refer to end users who will use the final software application to model prediction for operational decisions. As per Figure 5.3, domain experts utilize the visual interface, which utilizes AutoGeoML's API interfaces to incorporate their domain expertise into data pre-processing (i.e., windowing), feature selection, and model evaluation and interpretation.

• **Data.** It refers to the actual data used to train, update, and make predictions with the model. The domain expert validates the quality of the data (i.e., missing or abnormal data) and determines which data should be used for training or updating the model. In our scenario, the domain experts guarantee that the training and test datasets have similar characteristics (e.g., data frequency from sensors and units) or data windowing (e.g., the number of previous data points to be used for updating the model), etc.

• **Model interface.** It provides an abstraction by taking input datasets and predicting outputs based on the current objective defined by the user. Developing an ML model's high-level objective is to understand the model input and output relationship based on defined concepts. The users, i.e., domain experts in this situation, can directly influence the model (e.g., configuration of different parameters and hyper-parameters) or indirectly (e.g., choosing the dataset, model output). Our AutoGeoML provides a high-level flexible abstract interface to experiment with different ML algorithms. Our proposed library utilizes ensemble techniques to find the best ML algorithms for a training and test dataset. Domain experts interpret the model's predictions and can guide in choosing the best ML algorithms.

• **Visual interface.** It refers to the wider visual interface of the application that allows domain experts to select alternative sets of features and visually evaluate model predictions. This component is a bidirectional link between the end users and the ML model. Our AutoGeoML module is designed as a standalone component that can be incorporated with any visual interface module using programming APIs.

5.4.3 API description

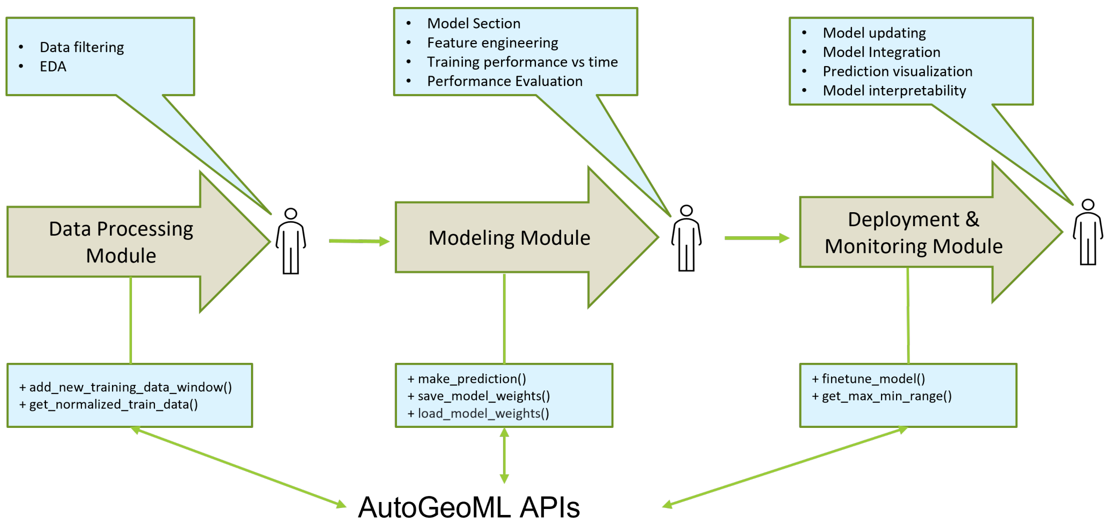
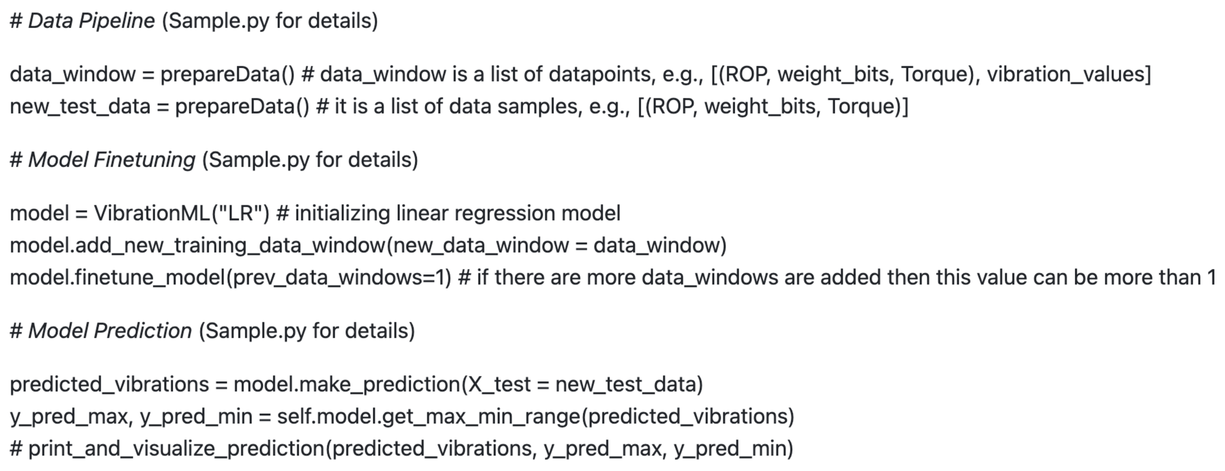


Figure (5.4) A high-level overview of AutoGeoML API modules for different tasks.

In this Section, we describe the implementation details of AutoGeoML, an open-source AutoML framework [32] for vibration detection for the drill string in horizontal drilling. It provides a modular end-to-end model API for model training, updating, and deployment. It basically consists of three modules: (1) Data Processing Module, and (2) Model Selection & Training Module, (3) Model Deployment & Monitoring Module. In Figure 5.5, we provide a high-level overview of these three modules with some of the AutoGeoML APIs to achieve various tasks as described in Section 5.3.2. We provide a brief overview of these modules below. The complete documentation of these APIs is available in our replication package.

• **Data Processing Module.** As described in Previous Section, the dataset in horizontal drilling is sequential. Thus



```
# Data Pipeline (Sample.py for details)

data_window = prepareData() # data_window is a list of datapoints, e.g., [(ROP, weight_bits, Torque), vibration_values]
new_test_data = prepareData() # it is a list of data samples, e.g., [(ROP, weight_bits, Torque)]

# Model Finetuning (Sample.py for details)

model = VibrationML("LR") # initializing linear regression model
model.add_new_training_data_window(new_data_window = data_window)
model.finetune_model(prev_data_windows=1) # if there are more data_windows are added then this value can be more than 1

# Model Prediction (Sample.py for details)

predicted_vibrations = model.make_prediction(X_test = new_test_data)
y_pred_max, y_pred_min = self.model.get_max_min_range(predicted_vibrations)
# print_and_visualize_prediction(predicted_vibrations, y_pred_max, y_pred_min)
```

Figure (5.5) A screenshot of sample API usage for AutoGeoML.

we need to store the sequence of this dataset for fine-tuning the model.

- add_new_training_data_window(new_data_window): As new drilling data becomes available in real-time, they are temporally stored inside a list called 'data_window,' and when there are sufficient data (e.g., 100, 200 or 600) the user, i.e., a domain expert can put them into the models' accessible data list for future fine-tuning.

- get_normalized_train_data(): This is a helper method that normalizes the training and test dataset using SkLearner's 'StandardScaler()' method.

• **Model Selection & Training Module.** This module provides a high-level abstraction over the design choices of the ML model and hyper-parameter selection. It also provides APIs for predicting on the test dataset and optional private methods to save and load the trained model's weight.

- VibrationML (model_type='best'): 'model_type' determines what type of model to develop and returns the best settings and hyperparameters: (1) 'LR': this initializes a linear regression model, (2) 'RF': initializes a for Random Forrest model, (3) 'LSTM': initializes an LSTM-based RNN model whose RNN architecture and hyperparameters worked best in our experiment settings, (4) 'best': by default, this initializes all these models but returns the best one after training,

- make_prediction (X_test): This API helps make real-time predictions on the previously trained model. Here 'X_test' is a list of test data points (e.g., a tuple of feature variables ('ROP', 'Torque', or 'WOB')).

- save_model_weights('mode_save_path'=None): After training, each model is stored at a predefined path. Based on the customization requirement we also provide methods 'load_model_weights(model_save_path=None)' to save and restore previously trained models. If the 'model_save_path' parameter is not provided, then a default path is used, defined at the time of model creation.

• **Model Deployment & Monitoring Module.** This module employs a modular approach to prediction and visualization. AutoGeoML does not include any visualization (e.g., data visualization or model prediction visualization) but provides a high-level, low-code API that is easily coupled with any visualization user interface. This module also gives domain experts a flexible API for updating models when new drilling data becomes available.

- finetune_model (prev_data_windows=1): This API is utilized for updating or refining the model as new drilling data becomes accessible. The default size of the preceding data window is 1, or the most recent few data points. However, based on our experiments, we discovered that, in some instances, model performance could be improved by adjusting the model based on the preceding few (e.g., 2 to 10) data windows (i.e., last 20 min to 2 hours of drilling.)
- get_max_min_range(): This API retrieves additional prediction information. This VibrationML is fundamentally a regression problem; hence, the model returns a single downhole vibration prediction value for a test unit. This API is utilized to easily retrieve a range of values (e.g., minimum and maximum range) for downhole vibration based on the user's specifications to detect anomalous behavior.

Figure 5.5 demonstrates a high-level overview with pseudocode code for initializing, training, and fine-tuning VibrationML. It demonstrates the modular design of the data input and model output pipeline. One can integrate this low-code vibration prediction library with any application and develop a vibration prediction model with as low as three lines of code.

5.5 Discussions

AutoML tools provide a high-level abstraction over multiple ML pipelines. They are making ML more accessible to domain experts and increasing the productivity of ML engineers. However, similar to other studies [110, 151, 271], we find that their fully automated off-the-self solution fails to consider the ways this technology is used in practice. For example, in our use cases, complete automation was not required; instead, only abstraction, i.e., automation over the model designing & training aspect, was desirable. Even in that cases, there was a requirement for model interpretability and transparency. In this Section, we discuss recommendations for future AutoML service providers based on our experience developing an industrial ML model.

- **Citizen practitioners with varying degrees of expertise.** AutoML tools target domain experts, novice ML engineers, and software developers and provide them with the tools required to develop a decent ML solution without being an ML expert. However, these practitioners have different objectives and skills based on the project. So, instead of focusing on a one-size-fits-all approach, future efforts can concentrate on designing considering the diverse comfort level of these diverse citizen practitioners.

- **Domain expertise and Model transparency.** Trust must be established between human users and ML systems. Without trust, attempts to automate several crucial aspects of the machine learning decision-making process for real-world use cases will be futile. The relationship between human interaction and automation is complex. In our circumstances, domain experts are familiar with the many features of the dataset; thus, delivering a black-box machine learning (ML) solution (e.g., abstracting feature engineering) does not establish trust and adoption of the AutoML solution. As stated by related research [271], we should seek a balance between the incorporation of human knowledge and the computational efficiency of hyperparameter tuning to eliminate biases and blind spots. We suggest that black box-like interfaces with low customizability and lack of transparency can hurt widespread AutoML adoption.

- **AutoML applications.** AutoML solutions are computationally intensive by nature. AutoML services, particularly AutoML cloud platforms, offer end-to-end ML workflow solutions for various application types and data sets. This enables novice individuals and companies to train computationally intensive ML models with minimal effort. However, most cloud service providers do not offer interoperability with external services. Consequently, cloud vs. non-cloud AutoML solutions support a distinct set of use cases and end users. Different domains, application types, and users require different levels of abstraction, customizability, transparency, and extensibility.

5.5.1 AutoML challenges during AutoGeoML development

In the previous Chapter 4, we present 13 AutoML-related topics organized into four categories based on an analysis of practitioners' discussions in SO. In this Section, we detail how many of these obstacles we encountered throughout the development of AutoGeoML to meet the nine functional and non-functional requirements presented in Table 5.1. Table 5.2 presents a summary of 13 AutoML topics we encountered during our development.

MLOps. For the development of AutoGeoML, we explored non-cloud-based AutoML frameworks because it provides us the necessary flexibility to train and deploy our model in our local environment (e.g., R2, R5, R8 in Table 5.1). It also enables us to integrate the ML solution with a larger software application. One trade-off is that we faced some issues related to Library Management (e.g., similar to library installation $Q_{68188449}$), Resource Management (e.g., GPU over-usage similar to $Q_{63656294}$), and Model deployment.

Model. It contains Model designing, debugging, training, and monitoring related topics. In our experience, we experienced that AutoML libraries provide decent performance with around ten-time computational costs. However, they lacked adequate debugging and community support. For the AutoGeoML project, modular design (i.e., R3) and model interpretability (i.e., R4) were important requirements that existing AutoML tools lacked. Instead of end-to-end automation rather, expert-in-the-loop design is more suitable for AutoGeoML.

Table (5.2) The summary observed AutoML related topics during AutoGeoML development

Topic Cat	Topic	Summary of challenges
MLOps	Library/Platform Management	Our studied non-cloud AutoML tools had proper support for installing and setting up the development environment as we conducted experiments on our machines. But our experience suggests that cloud-based AutoML solutions will be better suited for non-technical domain experts as they will be pre-configured with necessary libraries.
	Model Load & Deployment	Our studied AutoML tools had proper support for model loading and deployment to any cloud, but they lacked proper documentation to achieve that goal.
	Pipeline Automation	AutoML tools provide end-to-end automated pipelines from dataset to model training. But in our case, the requirement was to have better transparency over each ML pipeline and get domain experts' input at each step.
	Bot Development	This topic is irrelevant to us as this focus is on AutoML platforms' support to develop chatbots.
	Resource Management	This topic is regarding AutoML cloud solutions. At the same time, AutoML tools also provided enough information regarding how many resources it uses during model training.
Model	Model Performance	AutoML tools provided us with one of the best-performing ML models at the cost of more than ten times computational power.
	Model Training & Monitoring	AutoML tools provided decent APIs for this topic, and we faced no major obstacles.
	Model Debugging	AutoML tools severely lack debugging support. Their error messages are quite obscure and severely lack community support compared to more stable ML libraries such as TensorFlow.
	Model Design	AutoML tools provide good support for designing black box ML models. But in our case, we favored model transparency and interpretability over abstract design.
Data	Data Management	AutoML cloud solutions provide support for data management and storage solution. Non-cloud AutoML libraries usually lack these supports; thus, we designed a custom data management system.
	Data Transformation	Most of the AutoML tools focus on end-to-end automated pipelines and thus lack adequate data transformation APIs.
	Data Exploration	Most of the AutoML tools focus on end-to-end automated pipelines and thus lack customizable data exploration APIs.
Documentation	Library Documentation	We found that some of the AutoML tools lacked proper documentation. For example, the AutoKeras library provided good enough documentation for high-level AutoML APIs but lacked adequate documentation for pre-processing data APIs.

Data. Current AutoML libraries focus on the end-to-end automated pipeline, but for AutoGeoML (i.e., See 5.3.2), the requirement was to have a customizable data processing pipeline (e.g., R1, R6, and R7). During our experiments, we found the Data transformation and exploration APIs for AutoML tools lacking compared to other popular ML libraries such as Pandas, Sklearn, TensorFlow, etc.

Documentation. We find that non-cloud AutoML tools such as AutoKeras, and Auto-Sklearn provides decent documentation for their end-to-end pipeline. However, they lack adequate documentation for individual workflow steps (e.g., we encountered similar configuration-related issues $Q_{70781470}$).

5.6 Usefulness of the findings for Project Managers

Table (5.3) The summary of AutoML challenges observed in SO discussion vs AutoGeoML development

Theme	AutoML Challenges in SO Discussion	Challenges Observed During AutoGeoML Development	Summary of challenges
Library/Platform Management	Practitioners struggle with proper development environment, installing and upgrading packages.	We did not face any major challenges for setting up our development environment.	Our experience suggest that cloud-based AutoML solutions will be better suited for non-technical domain experts as they will be pre-configured with necessary libraries.
Data Transformation	Practitioners struggle with data encoding, filtering, processing large data frame.	We faced challenges to implement exploratory data analysis (R1) and data filtering features (R6)	Similar to SO discussion we find AutoML tools lack adequate data transformation APIs
Model Training & Monitoring	Practitioners struggle with training model, monitoring progress, improving performance	We did not face any major challenge in this regard.	Better NAS and hyper-parameter optimization technique can improve AutoML solution's performance.
Model Debugging	Practitioners struggle with debugging error messages	We also faced similar obscure error messages.	AutoML tools severely lack community support compared to more stable ML library such as TensorFlow.
Model Design	Practitioners struggle with transparency regarding feature engineering and model architecture.	We faced similar problem to develop transparent interpretable Model.	For industrial adoption AutoML requires better interpretable models.
Resource Management	Practitioners struggle with scaling up, GPU/CPU utilization	We also found AutoML tools use 10 to 100 times more computational power	Cloud-based AutoML solutions are better suited for on demand scaling.
Documentation	Practitioners struggle with inadequate and incomplete documentation	We also faced similar challenges.	Better tutorial-based documentation can greatly improve the AutoML adoption

Low-code ML toolkits aim to automate ML workflows so that domain experts may develop ML models more rapidly than with the conventional ML approach. The findings of this study can also help ML practitioners or project managers to get significant insights regarding choosing an AutoML solution. First of all, the project managers need to decide whether to use a cloud vs non-cloud AutoML solution for the particular project. From Section 4.4.4 managers can have a better understanding of the current strengths and limitations of cloud vs non-cloud AutoML solutions. For example, cloud solutions provide better end-to-end functionality whereas non-cloud solutions provide better customizability and modularity. Second, the project managers need to understand the tradeoff and potential pitfalls of choosing a low-code toolkit over a conventional ML toolkit. Our discussion (4.5.3 and 5.4) can help project managers to better understand the potential pitfalls and tradeoffs for using AutoML solutions. Our analysis also provides an in-depth understanding of current challenges (4.4.1), their distribution across ML life cycle phases (4.4.2), and their popularity and difficulty (4.4.3).

In addition, the approach and findings of this industrial case study can guide project managers to choose a popular AutoML toolkit based on their requirements. In Table 5.3, we present the summary of the AutoML challenges observed in SO discussion and during AutoGeoML development. We find that non-cloud AutoML tools still have limited support in terms of setting up development environment, data pre-processing, transparent model design, improving model's performance, adequate documentation and community support in SO. For example, we observe that AutoML tools provide black-box ML models by obscuring feature engineering and model architecture, and it consumes a disproportionately large amount of computing resources. During the course of our development and testing, we discovered that AutoML tools were producing obscure error messages with very little community support in SO. So, all these findings provide valuable insights for project managers to better understand the tradeoffs and better resource planning (i.e., human training, potential buggy APIs) and incorporate domain and ML experts into the projects.

5.7 Summary

In this study, we present the findings of our case study of developing a downhole vibration prediction model for horizontal drilling in collaboration with domain experts from industry and academia. We collect system requirements from real domain experts with operational experience in horizontal drilling. We face several challenges while developing the ML model, from data processing, Model designing, and ML operations. We also find that complete end-to-end automation is neither feasible nor desirable, as it requires a lot of flexibility in customization and interpretability to integrate into larger systems. Our study recommends automating each step of ML workflow and close collaboration with ML engineers and domain experts. We hope these findings and our design guidelines will help the researchers and vendors to develop the next generation of AutoML frameworks/services.

Chapter 6

Conclusion and Future Work

6.1 Summary

Low-code approach for traditional software applications is a novel paradigm for developing software applications using visual programming with minimum hand-coding. AutoML is a novel low-code approach for developing ML applications with minimal coding by utilizing higher-level end-to-end APIs. It automates various tasks in the ML pipeline, such as data prepossessing, model selection, hyperparameter tuning, etc.

We conduct a qualitative analysis of different aspects of 121 traditional LCSD services. We discover that 37% of LCSD platforms offer features for developing business process management (BPM)-related applications, followed by general-purpose apps (32%) and work process automation (25%). The 90% typical LCSD platforms are proprietary, and 63% of them offer a proprietary cloud deployment option (i.e., vendor lock-in) solely; 34% of platforms offer both a third-party deployment option alongside vendor cloud deployment. Approximately 98% of LCSD platforms exclusively support web-based application development. We perform a qualitative analysis of 37 AutoML vendors. Among these vendors, 43% offer Shallow ML, 22% general ML, 19% data analytics, and 16% deep learning application development services, respectively. 51% of overall AutoML services are non-cloud-based tools (i.e., libraries/frameworks). We find that 57% of AutoML solutions are open-source and that among these open-source services, 52% are from the tech industry and 48% are from academia, underscoring the significant participation of academic and industrial researchers.

We present an empirical study that sheds light on the types of discussions low-code developers discuss in Stack Overflow (SO). We find 40 low-code topics for traditional applications in our dataset of 33.7K SO posts (question + accepted answers). We divide them into five high-level groups, namely Application Customization (30% Questions, 11 Topics), Data Storage (25% Questions, 9 Topics), Platform Adoption (20% Questions, 9 Topics), Platform

Maintenance (14% Questions, 6 Topics), and Third-Party Integration (12% Questions, 5 Topics). We find that the Platform Adoption topic category has gained popularity recently. Platform Related Query and Message Queue topics from this category are the most popular. On the other hand, we also find that practitioners find Platform Adoption and Maintenance-related topics most difficult. The most frequent queries are of the how-type, but our study shows that what-type and why-type questions are more challenging for practitioners. Despite extensive testing, deployment, and maintenance support, our analysis shows that server configuration, data migration, and system module upgrading-related queries are widespread and complex to LCSD practitioners.

We find 13 AutoML topics from our dataset of 14.3K extracted SO posts (question + acc. answers). We divide them into four high-level categories, namely the MLOps category (5 topics, 43.2% questions) with the highest number of SO questions, followed by Model (4 topics, 27.6% questions), Data (3 topics, 27% questions), Documentation (1 topic, 2.2% questions). Despite widespread support for data management, model design & deployment, we see that these concerns continue to dominate discussions among AutoML practitioners throughout various MLLC stages. We discover that many inexperienced practitioners have unresolved platform feature-related questions.

For our case study, which involved working with experts from both industry and academia, we developed a downhole vibration prediction model for horizontal drilling. We compile system requirements from actual subject-matter experts with practical knowledge. We face the majority of the challenges as we outlined in Chapter 4 while developing the ML model, from data processing, Model designing, and ML operations. In addition, we discover that full end-to-end automation is neither practical nor desired because it requires a great deal of customization and interpretability to be integrated into bigger systems.

Our analysis indicates that many of these issues can be resolved using better tutorial-based documentation. To address the many LCSD difficulties, it is our intention that all of these findings would assist different LCSD stakeholders (such as LCSD platform suppliers, practitioners, and SE researchers). According to studies and the popularity of LCSD developers, many organizations are likely to use this innovation for consumer-facing applications or to generate business insight from their dataset. For LCSD to be quickly adopted, researchers and service providers must overcome the issues facing developers. Our case study suggests closely working with ML engineers and domain experts and automating every step of the ML workflow. We hope the researchers and vendors will use these findings and our design recommendations to develop future LCSD frameworks and services.

6.2 Limitations and Future Work

Future work can concentrate on the following limitations. In our qualitative assessment of the current status of low-code services, we analyzed 158 low-code solution providers for traditional and ML application development. We find that around 20% of them are open-sourced, and most of them are deployed in GitHub or other public repositories.

In our future research effort, we can explore the development activities such as commits, issue count, rating, and the number of forks of traditional vs. ML low-code tools/frameworks.

To understand their challenges, we reviewed low-code practitioners' discussions in SO for this research. SO is a large online technical Q&A forum with around 120 million posts and 12 million registered users [188]. We anticipate that SO will contain more technical-related discussions. But we chose SO because it gives us good generalizability and contains general discussions from software engineers and people with other disciples worldwide. Every month, around 50 million people from varied backgrounds, including students, novices, professionals, educators, and academic researchers in various career areas, such as full-stack, backend, mobile developers, sales professionals, etc., visit Stack Overflow. However, we also believe this study can be complemented by including discussions from other forums, surveying, and interviewing low-code developers.

We deduce conclusions and suggestions for our empirical research based on our observation of practitioners discussing in SO. Consequently, further validation from the developer survey might give more information. Nevertheless, the variety of low-code platforms and themes makes it difficult to build a representative survey sample of LCSD practitioners. The data may thus be utilized to construct multiple LCSD and AutoML-related surveys concentrating on various low-code topics or specific platforms.

Our future work will focus on (1) getting developers' feedback on our study findings based on surveys and developer interviews, (2) developing tools and techniques to address the challenges in the LCSD platforms we observed automatically, (3) developing tools to address the issues observed in the existing AutoML's data processing and model designing pipeline, and (4) by conducting in-depth interviews and surveys with relevant stakeholders on the efficacy of our proposed AutoGeoML framework

Bibliography

[1] Google Introduces Cloud AutoML In A Bid To Democratise Artificial Intelligence. `https://analyticsindiamag.com/google-cloud-automl-vision-democratising-ai/`. [Online; accessed 5-December-2022].

[2] A. Abdellatif, D. Costa, K. Badran, R. Abdalkareem, and E. Shihab. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 174–185, New York, NY, USA, 2020. Association for Computing Machinery.

[3] B. Adrian, S. Hinrichsen, and A. Nikolenko. App development via low-code programming as part of modern industrial engineering education. In *International Conference on Applied Human Factors and Ergonomics*, pages 45–51. Springer, 2020.

[4] E. Aghajani, C. Nagy, M. Linares-Vásquez, L. Moreno, G. Bavota, M. Lanza, and D. C. Shepherd. Software documentation: the practitioners' perspective. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 590–601. IEEE, 2020.

[5] A. Agrapetidou, P. Charonyktakis, P. Gogas, T. Papadimitriou, and I. Tsamardinos. An automl application to forecasting bank failures. *Applied Economics Letters*, 28(1):5–9, 2021.

[6] A. Agrawal, W. Fu, and T. Menzies. What is wrong with topic modeling? and how to fix it using search-based software engineering. *Information and Software Technology*, 98:74–88, 2018.

[7] S. Ahmed and M. Bagherzadeh. What do concurrency developers ask about? a large-scale study using stack overflow. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery.

[8] P. A. Akiki, P. A. Akiki, A. K. Bandara, and Y. Yu. Eud-mars: End-user development of model-driven adaptive robotics software systems. *Science of Computer Programming*, 200:102534, 2020.

[9] M. A. Al Alamin and G. Uddin. Quality assurance challenges for machine learning software applications during software development life cycle phases. In *2021 IEEE International Conference on Autonomous Systems (ICAS)*, pages 1–5. IEEE, 2021.

[10] M. A. A. Alamin, S. Malakar, G. Uddin, S. Afroz, T. B. Haider, and A. Iqbal. An empirical study of developer discussions on low-code software development challenges. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 46–57. IEEE, 2021.

[11] M. A. A. Alamin, G. Uddin, S. Malakar, S. Afroz, T. Haider, and A. Iqbal. Developer discussion topics on the adoption and barriers of low code software development platforms. *Empirical Software Engineering*, 28(1):1–59, 2023.

[12] A. N. Alonso, J. Abreu, D. Nunes, A. Vieira, L. Santos, T. Soares, and J. Pereira. Towards a polyglot data access layer for a low-code application development platform. *arXiv preprint arXiv:2004.13495*, 2020.

[13] H. A. ALSAADI, D. T. RADAIN, M. M. ALZAHRANI, W. F. ALSHAMMARI, D. ALAHMADI, and B. FAKIEH. Factors that affect the utilization of low-code development platforms: survey study. *Romanian Journal of Information Technology and Automatic Control*, 31(3):123–140, 2021.

[14] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu. Why is developing machine learning applications challenging? a study on stack overflow posts. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE, 2019.

[15] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[16] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. Power to the people: The role of humans in interactive machine learning. *Ai Magazine*, 35(4):105–120, 2014.

[17] AWS Amplify Studio overview. Available: `https://aws.amazon.com/amplify/studio/`. [Online; accessed 5-January-2022].

[18] Oracle Apex Platform. Available: `https://apex.oracle.com/`. [Online; accessed 5-January-2022].

[19] App Engine: a fully managed, serverless platform for developing and hosting web applications at scale. Available: `https://cloud.google.com/appengine/docs`. [Online; accessed 13-December-2021].

[20] Appery platform overview. Available: `https://appery.io/`. [Online; accessed 5-January-2022].

[21] Appian platform overview. Available: `https://www.appian.com/`. [Online; accessed 5-January-2021].

[22] R. Arun, V. Suresh, C. V. Madhavan, and M. N. Murthy. On finding the natural number of topics with latent dirichlet allocation: Some observations. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 391–402. Springer, 2010.

[23] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider. Answering questions about unanswered questions of stack overflow. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 97–100. IEEE, 2013.

[24] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 95–104. IEEE, 2010.

[25] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE software*, 20(5):36–41, 2003.

[26] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[27] AutoFolio Automated Algorithm Selection with Hyperparameter Optimization Library. Available: https://github.com/automl/AutoFolio. [Online; accessed 5-January-2022].

[28] Automated machine learning for analytics & production. Available: https://github.com/ClimbsRocks/auto_ml. [Online; accessed 5-January-2022].

[29] Google's AutoML: Cutting Through the Hype . Available: https://www.fast.ai/2018/07/23/auto-ml-3/. [Online; accessed 5-January-2021].

[30] Automated Machine Learning Market Research Report - Global Industry Analysis and Growth Forecast to 2030. Available: https://www.researchandmarkets.com/reports/5010695/automated-machine-learning-market-research-report. [Online; accessed 5-January-2022].

[31] AutoML: How to Automate Machine Learning With Google Vertex AI, Amazon SageMaker, H20.ai, and Other Providers. Available: https://www.altexsoft.com/blog/automl/. [Online; accessed 5-January-2022].

[32] Code for AutoGeoML Library. Available: https://github.com/disa-lab/Thesis-Alamin-MSc/tree/main/Chapter_5. [Online; accessed 5-December-2022].

[33] Amazon Lex - Conversational AI and Chatbots. Available: https://aws.amazon.com/lex/. [Online; accessed 5-January-2022].

[34] AWS Announces Nine New Amazon SageMaker Capabilities. Available: https://www.businesswire.com/news/home/20201208005335/en/AWS-Announces-Nine-New-Amazon-SageMaker-Capabilities. [Online; accessed 5-January-2022].

[35] Amazon SageMaker Overview. Available: https://aws.amazon.com/sagemaker/. [Online; accessed 5-January-2022].

[36] Azure Machine Learning - ML as a Service. Available: https://azure.microsoft.com/en-us/services/machine-learning/. [Online; accessed 5-January-2022].

[37] M. Bagherzadeh and R. Khatchadourian. Going big: A large-scale study on what big data developers ask. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 432–442, New York, NY, USA, 2019. ACM.

[38] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435*, 2015.

[39] K. Bajaj, K. Pattabiraman, and A. Mesbah. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 112–121, 2014.

[40] A. Baltadzhieva and G. Chrupała. Predicting the quality of questions on stackoverflow. In *Proceedings of the international conference recent advances in natural language processing*, pages 32–40, 2015.

[41] A. Bandeira, C. A. Medeiros, M. Paixao, and P. H. Maia. We need to talk about microservices: an analysis from the discussions on stackoverflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 255–259. IEEE, 2019.

[42] A. A. Bangash, H. Sahar, S. Chowdhury, A. W. Wong, A. Hindle, and K. Ali. What do developers know about machine learning: a study of ml discussions on stackoverflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 260–264. IEEE, 2019.

[43] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

[44] F. Basciani, L. Iovino, A. Pierantonio, et al. Mdeforge: an extensible web-based modeling platform. In *2nd International Workshop on Model-Driven Engineering on and for the Cloud, CloudMDE 2014, Co-located with the 17th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2014*, volume 1242, pages 66–75. CEUR-WS, 2014.

[45] V. R. Basil and A. J. Turner. Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, (4):390–396, 1975.

[46] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, and A. D. Lucia. Methodbook: Recommending move method refactorings via relational topic models. *IEEE Transactions on Software Engineering*, 40(7):671–694, 2014.

[47] J. Bayer and D. Muthig. A view-based approach for improving software documentation practices. In *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)*, pages 10–pp. IEEE, 2006.

[48] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. 2001.

[49] P. Beynon-Davies, C. Carne, H. Mackay, and D. Tudhope. Rapid application development (rad): an empirical review. *European Journal of Information Systems*, 8(3):211–223, 1999.

[50] J. M. Bhat, M. Gupta, and S. N. Murthy. Overcoming requirements engineering challenges: Lessons from offshore outsourcing. *IEEE software*, 23(5):38–44, 2006.

[51] E. Bisong. Google automl: cloud vision. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 581–598. Springer, 2019.

[52] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.

[53] D. G. Bonett and T. A. Wright. Sample size requirements for estimating pearson, kendall and spearman correlations. *Psychometrika*, 65(1):23–28, 2000.

[54] M. Boshernitsan and M. S. Downes. *Visual programming languages: A survey*. Citeseer, 2004.

[55] G. Botterweck. A model-driven approach to the engineering of multiple user interfaces. In *International Conference on Model Driven Engineering Languages and Systems*, pages 106–115. Springer, 2006.

[56] Workflow Automation Software Explained, and How to Provide a Visual Integration Builder to Your Clients. Available: `https://www.appmixer.com/blog/workflow-automation-software-explained`. [Online; accessed 5-January-2022].

[57] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

[58] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

[59] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[60] M. M. Burnett and D. W. McIntyre. Visual programming. *COMPUTER-LOS ALAMITOS-*, 28:14–14, 1995.

[61] P. Chakraborty, R. Shahriyar, A. Iqbal, and G. Uddin. How do developers discuss and support new programming languages in technical q&a site? an empirical study of go, swift, and rust in stack overflow. *Information and Software Technology (IST)*, page 19, 2021.

[62] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[63] T.-H. Chen, S. W. Thomas, M. Nagappan, and A. E. Hassan. Explaining software defects using topic models. In *9th working conference on mining software repositories*, pages 189–198, 2012.

[64] T.-H. P. Chen, S. W. Thomas, and A. E. Hassan. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, 21(5):1843–1919, 2016.

[65] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu. A comprehensive study on challenges in deploying deep learning based software. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 750–762, 2020.

[66] G. G. Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.

[67] B. Cleary, C. Exton, J. Buckley, and M. English. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, 14:93–130, 2009.

[68] T. J. Cleophas and A. H. Zwinderman. Bayesian pearson correlation analysis. In *Modern Bayesian Statistics in Clinical Research*, pages 111–118. Springer, 2018.

[69] M. F. Costabile, D. Fogli, P. Mussio, and A. Piccinno. Visual interactive systems for end-user development: a model-based design methodology. *IEEE transactions on systems, man, and cybernetics-part a: systems and humans*, 37(6):1029–1046, 2007.

[70] A. Cummaudo, R. Vasa, S. Barnett, J. Grundy, and M. Abdelrazek. Interpreting cloud computer vision pain-points: A mining study of stack overflow. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1584–1596. IEEE, 2020.

[71] D. Dahlberg. Developer experience of a low-code platform: An exploratory study, 2020.

[72] Darwin - Automated Machine Learning Platform. Available: `https://www.sparkcognition.com/product/darwin/`. [Online; accessed 5-January-2022].

[73] K. Das and R. N. Behera. A survey on machine learning: concept, algorithms and applications. *International Journal of Innovative Research in Computer and Communication Engineering*, 5(2):1301–1309, 2017.

[74] DataRobot AI Cloud - The Next Generation of AI. Available: `https://www.datarobot.com/`. [Online; accessed 5-January-2022].

[75] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Labeling source code with information retrieval methods: an empirical study. *Empirical Software Engineering*, 19(5):1383–1420, 2014.

[76] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[77] C. Di Sipio, D. Di Ruscio, and P. T. Nguyen. Democratizing the development of recommender systems by means of low-code platforms. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–9, 2020.

[78] J. Dieber and S. Kirrane. Why model why? assessing the strengths and limitations of lime. *arXiv preprint arXiv:2012.00093*, 2020.

[79] J. J. Dudley and P. O. Kristensson. A review of user interface design for interactive machine learning. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 8(2):1–37, 2018.

[80] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[81] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[82] A. et al. Software engineering challenges of deep learning. In *proc of Software Engineering and Advanced Applications (SEAA)*, pages 50–59. IEEE, 2018.

[83] A. et al. Software engineering for machine learning: A case study. In *proc ICSE-SEIP*, pages 291–300. IEEE, 2019.

[84] B. et al. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2014.

[85] D. et al. Deep learning & software engineering: State of research and future directions. *arXiv preprint arXiv:2009.08525*, 2020.

[86] F. et al. Ways of applying artificial intelligence in software engineering. In *proc RAISE*, pages 35–41. IEEE, 2018.

[87] G. et al. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[88] J. et al. To trust or not to trust a classifier. In *proc NeurIPS*, pages 5546–5557, 2018.

[89] Q. et al. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, 9(5):909, 2019.

[90] R. et al. Explainable machine learning for scientific insights and discoveries. *IEEE Access*, 8:42200–42216, 2020.

[91] S. et al. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511, 2015.

[92] S. et al. Tutorial: safe and reliable machine learning. *arXiv preprint arXiv:1904.07204*, 2019.

[93] S. et al. Machine learning for software engineering: A systematic mapping. *arXiv preprint arXiv:2005.13299*, 2020.

[94] W. et al. How does machine learning change software development practices? *In TSE*, 2019.

[95] Z. et al. Machine learning testing: Survey, landscapes and horizons. *In TSE*, 2020.

[96] S. et al.r. Engineering reliable deep learning systems. *arXiv preprint arXiv:1910.12582*, 2019.

[97] S. Exchange. Stack exchange data dump . Available: `https://archive.org/details/stackexchange`, 2020. [Online; accessed 5-January-2022].

[98] J. A. Fails and D. R. Olsen Jr. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 39–45, 2003.

[99] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. 2020.

[100] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: The next generation. *arXiv preprint arXiv:2007.04074*, 2020.

[101] R. Fiebrink, P. R. Cook, and D. Trueman. Human model evaluation in interactive supervised learning. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 147–156, 2011.

[102] S. Fincher and J. Tenenberg. Making sense of card sorting data. *Expert Systems*, 22(3):89–93, 2005.

[103] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev. Meta-design: a manifesto for end-user development. *Communications of the ACM*, 47(9):33–37, 2004.

[104] N. Fors. *The Design and Implementation of Bloqqi-A Feature-Based Diagram Programming Language*. PhD thesis, Lund University, 2016.

[105] M. Fryling. Low code app development. *J. Comput. Sci. Coll.*, 34(6):119, Apr. 2019.

[106] G2 overview. Available: `https://www.g2.com/`. [Online; accessed 5-January-2022].

[107] Enterprise Low-Code Application Platforms (LCAP) Reviews and Ratings. Available: `https://www.gartner.com/reviews/market/enterprise-low-code-application-platform`. [Online; accessed 5-January-2022].

[108] Garner overview. Available: `https://www.gartner.com`. [Online; accessed 5-January-2022].

[109] P. Gijsbers, E. LeDell, J. Thomas, S. Poirier, B. Bischl, and J. Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[110] Y. Gil, J. Honaker, S. Gupta, Y. Ma, V. D'Orazio, D. Garijo, S. Gadewar, Q. Yang, and N. Jahanshad. Towards human-guided machine learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 614–624, 2019.

[111] Your AI pair programmer. Available: `https://github.com/features/copilot`. [Online; accessed 5-January-2021].

[112] AppSheet, Low-code application development. Available: https://www.appsheet.com. [Online; accessed 13-December-2021].

[113] Google App Maker platform overview. Available: https://developers.google.com/appmaker. [Online; accessed 5-January-2021].

[114] Cloud AutoML Custom Machine Learning Models. Available: https://cloud.google.com/automl. [Online; accessed 5-January-2022].

[115] Google cloud automl. Available: https://www.cloud.google.com/automl/. [Online; accessed 5-January-2022].

[116] Google App Maker will be shut down on January 19, 2021. https://workspaceupdates.googleblog.com/2020/01/app-maker-update.html. [Online; accessed 5-January-2021].

[117] A. Gulli and S. Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[118] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 810–822. IEEE, 2019.

[119] H2O.ai: AI Cloud Platform. Available: https://h2o.ai/. [Online; accessed 5-January-2022].

[120] H2O Driverless AI. Available: https://h2o.ai/products/h2o-driverless-ai/. [Online; accessed 5-January-2022].

[121] B. Hailpern and P. Tarr. Model-driven development: The good, the bad, and the ugly. *IBM systems journal*, 45(3):451–461, 2006.

[122] D. C. Halbert. *Programming by example*. PhD thesis, University of California, Berkeley, 1984.

[123] M. Hammer. What is business process management? In *Handbook on business process management 1*, pages 3–16. Springer, 2015.

[124] J. Han, E. Shihab, Z. Wan, S. Deng, and X. Xia. What do programmers discuss about deep learning frameworks. *Empirical Software Engineering*, 25(4):2694–2747, 2020.

[125] M. U. Haque, L. H. Iwaya, and M. A. Babar. Challenges in docker development: A large-scale study using stack overflow. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2020.

[126] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[127] R. R. Hoffman, S. T. Mueller, G. Klein, and J. Litman. Metrics for explainable ai: Challenges and prospects. *arXiv preprint arXiv:1812.04608*, 2018.

[128] Amazon Honeycode platform overview. Available: https://www.honeycode.aws/. [Online; accessed 5-January-2022].

[129] J. Hu, X. Sun, D. Lo, and B. Li. Modeling the evolution of development topics using dynamic topic models. In *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering*, pages 3–12, 2015.

[130] S. Hu, S. Xie, H. Zheng, C. Liu, J. Shi, X. Liu, and D. Lin. Dsnas: Direct neural architecture search without parameter retraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12084–12092, 2020.

[131] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1110–1121, 2020.

[132] F. Ihirwe, D. Di Ruscio, S. Mazzini, P. Pierini, and A. Pierantonio. Low-code engineering for internet of things: A state of research. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–8, 2020.

[133] F. Ihirwe, D. Di Ruscio, S. Mazzini, P. Pierini, and A. Pierantonio. Low-code engineering for internet of things: A state of research. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[134] M. J. Islam, H. A. Nguyen, R. Pan, and H. Rajan. What do developers ask about ml libraries? a large-scale study using stack overflow. *arXiv preprint arXiv:1906.11940*, 2019.

[135] A. Jacinto, M. Lourenço, and C. Ferreira. Test mocks for low-code applications built with outsystems. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–5, 2020.

[136] H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.

[137] joget low-code platform. Available: `https://github.com/jogetworkflow/jw-community`. [Online; accessed 5-January-2021].

[138] What is Joget? Available: `https://www.joget.org/product/`. [Online; accessed 5-January-2022].

[139] C. Jones. End user programming. *Computer*, 28(9):68–70, 1995.

[140] S. K. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni. Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)*, 54(8):1–36, 2021.

[141] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1):81–93, 1938.

[142] J. Y. Khan, M. T. I. Khondaker, G. Uddin, and A. Iqbal. Automatic detection of five api documentation smells: Practitioners' perspectives. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 318–329. IEEE, 2021.

[143] J. Y. Khan, M. T. I. Khondaker, G. Uddin, and A. Iqbal. Automatic detection of five api documentation smells: Practitioners' perspectives. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, page 12, 2021.

[144] F. Khorram, J.-M. Mottu, and G. Sunyé. Challenges & opportunities in low-code testing. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–10, 2020.

[145] F. Khorram, J.-M. Mottu, and G. Sunyé. Challenges & opportunities in low-code testing. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[146] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka: Automatic model selection and hyper-parameter optimization in weka. In *Automated machine learning*, pages 81–95. Springer, Cham, 2019.

[147] P. Kourouklidis, D. Kolovos, N. Matragkas, and J. Noppen. Towards a low-code solution for monitoring machine learning model performance. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–8, 2020.

[148] W. H. Kruskal. Historical notes on the wilcoxon unpaired two-sample test. *Journal of the American Statistical Association*, 52(279):356–360, 1957.

[149] Software Developers and Engineers Shortage in the US: The case for Global Talent. Available: `https://www.revelo.com/blog/software-developer-shortage-us`. [Online; accessed 5-January-2022].

[150] E. LeDell and S. Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020, 2020.

[151] D. J.-L. Lee and S. Macke. A human-in-the-loop perspective on automl: Milestones and the road ahead. *IEEE Data Engineering Bulletin*, 2020.

[152] T. C. Lethbridge. Low-code is often high-code, so we must design low-code platforms to enable proper software engineering. In *International Symposium on Leveraging Applications of Formal Methods*, pages 202–212. Springer, 2021.

[153] C. Li, X. Yuan, C. Lin, M. Guo, W. Wu, J. Yan, and W. Ouyang. Am-lfs: Automl for loss function search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8410–8419, 2019.

[154] H. Li, T.-H. P. Chen, W. Shang, and A. E. Hassan. Studying software logging using topic models. *Empirical Software Engineering*, 23:2655–2694, 2018.

[155] Y. Li, Y. Shen, W. Zhang, C. Zhang, and B. Cui. Volcanoml: speeding up end-to-end automl via scalable search space decomposition. *The VLDB Journal*, pages 1–25, 2022.

[156] Y. Li, Z. Wang, Y. Xie, B. Ding, K. Zeng, and C. Zhang. Automl: From methodology to application. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4853–4856, 2021.

[157] Lianja App Builder platform overview. Available: `https://www.lianja.com/overview/lianja-app-builder`. [Online; accessed 5-January-2022].

[158] O. Lieber, O. Sharir, B. Lenz, and Y. Shoham. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*, 2021.

[159] H. Lieberman, F. Paternò, M. Klann, and V. Wulf. End-user development: An emerging paradigm. In *End user development*, pages 1–8. Springer, 2006.

[160] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang. Software vulnerability detection using deep neural networks: a survey. *Proceedings of the IEEE*, 108(10):1825–1848, 2020.

[161] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk. An exploratory analysis of mobile development issues using stack overflow. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 93–96. IEEE, 2013.

[162] E. Loper and S. Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[163] IBM Lotus software. Available: `https://help.hcltechsw.com/`. [Online; accessed 5-January-2022].

[164] Low-code development platform . Available: `https://en.wikipedia.org/wiki/Low-code_development_platform`. [Online; accessed 5-January-2021].

[165] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan. Characteristics and challenges of low-code development: The practitioners' perspective. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2021.

[166] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In *International Conference on Agile Software Development*, pages 227–243. Springer, Cham, 2019.

[167] H. Mazzawi, X. Gonzalvo, A. Kracun, P. Sridhar, N. A. Subrahmanya, I. Lopez-Moreno, H. jin Park, and P. Violette. Improving keyword spotting and language identification via neural architecture search at scale. In *INTERSPEECH*, 2019.

[168] A. K. McCallum. Mallet: A machine learning for language toolkit. *http://mallet. cs. umass. edu*, 2002.

[169] M. L. McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.

[170] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, pages 7588–7598. PMLR, 2021.

[171] Mendix platform overview. Available: `https://www.mendix.com/`. [Online; accessed 5-January-2021].

[172] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[173] From conversation to code: Microsoft introduces its first product features powered by GPT-3. Available: `https://blogs.microsoft.com/ai/from-conversation-to-code-microsoft-introduces-its-first-product-feature` [Online; accessed 5-January-2021].

[174] Microsoft Power FX. Available: `https://docs.microsoft.com/en-us/power-platform/power-fx/overview`. [Online; accessed 13-December-2021].

[175] R. Mihalcea, H. Liu, and H. Lieberman. Nlp (natural language processing) for nlp (natural language programming). In *International Conference on intelligent text processing and computational linguistics*, pages 319–330. Springer, 2006.

[176] S. Mirjalili. Genetic algorithm. In *Evolutionary algorithms and neural networks*, pages 43–55. Springer, 2019.

[177] MLBox platform overview. Available: `https://bigml.com/`. [Online; accessed 5-January-2022].

[178] MLOps Overview. Available: `https://ml-ops.org/`. [Online; accessed 5-January-2022].

[179] B. A. Myers, A. J. Ko, and M. M. Burnett. Invited research overview: end-user programming. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 75–80, 2006.

[180] P. Nagarnaik and A. Thomas. Survey on recommendation system methods. In *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, pages 1603–1608. IEEE, 2015.

[181] C. Ness and M. E. Hansen. Potential of low-code in the healthcare sector: an exploratory study of the potential of low-code development in the healthcare sector in norway. Master's thesis, 2019.

[182] P. Okoli, J. Cruz Vega, and R. Shor. Estimating downhole vibration via machine learning techniques using only surface drilling parameters. In *SPE Western Regional Meeting*. OnePetro, 2019.

[183] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[184] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the genetic and evolutionary computation conference 2016*, pages 485–492, 2016.

[185] OneBlink platform overview. Available: `https://www.oneblink.io/`. [Online; accessed 5-January-2022].

[186] J. P. Ono, S. Castelo, R. Lopez, E. Bertini, J. Freire, and C. Silva. Pipelineprofiler: A visual analytics tool for the exploration of automl pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):390–400, 2020.

[187] M. Overeem and S. Jansen. Proposing a framework for impact analysis for low-code development platforms. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 88–97. IEEE, 2021.

[188] S. Overflow. *Stack Overflow Questions*. `https://stackoverflow.com/questions/`, 2020. Last accessed on 14 November 2020.

[189] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh. Deep learning vs. traditional computer vision. In *Science and Information Conference*, pages 128–144. Springer, 2019.

[190] Programming Gains Speed As Developers Turn to Low-Code During the Pandemic. Available: `https://www.designnews.com/automation/programming-gains-speed-developers-turn-low-code-during-pandemic`. [Online; accessed 5-August-2022].

[191] J. Pane and B. Myers. *More Natural Programming Languages and Environments*, pages 31–50. Springer, 10 2006.

[192] K. Patel, J. Fogarty, J. A. Landay, and B. L. Harrison. Examining difficulties software developers encounter in the adoption of statistical machine learning. In *AAAI*, pages 1563–1566, 2008.

[193] F. Paternò. End user development: Survey of an emerging field for empowering people. *International Scholarly Research Notices*, 2013, 2013.

[194] The Best Low-Code Development Platforms. Available: `https://www.pcmag.com/picks/the-best-low-code-development-platforms`. [Online; accessed 5-January-2021].

[195] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[196] J. Peppard. Customer relationship management (crm) in financial services. *European Management Journal*, 18(3):312–327, 2000.

[197] V. S. Phalake and S. D. Joshi. Low code development platform for digital transformation. In *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*, pages 689–697. Springer, 2021.

[198] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[199] A. Pleuss, S. Wollny, and G. Botterweck. Model-driven development and evolution of customized user interfaces. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 13–22, 2013.

[200] A. Płońska and P. Płoński. Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3, 2021.

[201] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. Improving low quality stack overflow post detection. In *2014 IEEE international conference on software maintenance and evolution*, pages 541–544. IEEE, 2014.

[202] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. T. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, 33(6):420–432, 2007.

[203] Microsoft power apps platform overview. Available: `https://powerapps.microsoft.com/en-us/`. [Online; accessed 5-January-2021].

[204] Quickbase platform overview. Available: `https://www.quickbase.com/product/product-overview`. [Online; accessed 5-January-2021].

[205] C. Ramasubramanian and R. Ramya. Effective pre-processing activities in text mining using improved porter's stemming algorithm. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(12):4536–4538, 2013.

[206] S. Rao and A. C. Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *8th Working Conference on Mining Software Repositories*, page 43–52, 2011.

[207] RapidMiner: Amplify the Impact of Your People, Expertise. Available: `https://rapidminer.com/`. [Online; accessed 5-January-2022].

[208] R. Rehurek and P. Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[209] X. Ren, Z. Xing, X. Xia, G. Li, and J. Sun. Discovering, explaining and summarizing controversial discussions in community q&a sites. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 151–162. IEEE, 2019.

[210] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[211] Rintagi low-code platform. Available: `https://github.com/Rintagi/Low-Code-Development-Platform`. [Online; accessed 5-January-2021].

[212] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford. Automated API property inference techniques. *IEEE Transactions on Software Engineering*, page 28, 2012.

[213] M. Röder, A. Both, and A. Hinneburg. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*, pages 399–408, 2015.

[214] C. Rosen and E. Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, page 33, 2015.

[215] C. Rosen and E. Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[216] J. R. Rymer, R. Koplowitz, and S. A. Leaders. The forrester wave(tm) low-code development platforms for ad&d professionals, q1 2019. 2019.

[217] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178. IEEE, 2020.

[218] Salesforce platform overview. Available: `https://www.salesforce.com/in/?ir=1`. [Online; accessed 5-January-2021].

[219] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28, 2015.

[220] B. Selic. The pragmatics of model-driven development. *IEEE software*, 20(5):19–25, 2003.

[221] B. Selic. A systematic approach to domain-specific language design using uml. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 2–9. IEEE, 2007.

[222] V. Shah, J. Lacanlale, P. Kumar, K. Yang, and A. Kumar. Towards benchmarking feature type inference for automl platforms. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1584–1596, 2021.

[223] P. Singh and A. Manure. Introduction to tensorflow 2.0. In *Learn TensorFlow 2.0*, pages 1–24. Springer, 2020.

[224] G. Sinha, R. Shahi, and M. Shankar. Human computer interaction. In *2010 3rd International Conference on Emerging Trends in Engineering and Technology*, pages 1–4. IEEE, 2010.

[225] SKLearn ML library. Available: `https://scikit-learn.org/stable/`. [Online; accessed 5-January-2022].

[226] Skyve low-code platform overview. Available: https://skyve.org/. [Online; accessed 5-January-2021].

[227] Software development overview. Available: https://en.wikipedia.org/wiki/Software_development. [Online; accessed 5-January-2021].

[228] Splunk: The Data Platform for the Hybrid World. Available: https://www.splunk.com/. [Online; accessed 5-January-2022].

[229] Struct platform overview. Available: https://structr.com/. [Online; accessed 5-January-2022].

[230] X. Sun, B. Li, H. Leung, B. Li, and Y. Li. Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks. *Information and Software Technology*, 66:671–694, 2015.

[231] X. Sun, B. Li, Y. Li, and Y. Chen. What information in software historical repositories do we need to support software maintenance tasks? an approach based on topic model. *Computer and Information Science*, pages 22–37, 2015.

[232] X. Sun, X. Liu, B. Li, Y. Duan, H. Yang, and J. Hu. Exploring topic models in software engineering data analysis: A survey. In *17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 357–362, 2016.

[233] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[234] A. Team. Azureml: Anatomy of a machine learning service. In *Conference on Predictive APIs and Apps*, pages 1–13. PMLR, 2016.

[235] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. Modeling the evolution of topics in source code histories. In *8th working conference on mining software repositories*, pages 173–182, 2011.

[236] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. Studying software evolution using topic models. *Science of Computer Programming*, 80(B):457–479, 2014.

[237] K. Tian, M. Revelle, and D. Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *6th international working conference on mining software repositories*, pages 163–166, 2009.

[238] C. Torres. Demand for programmers hits full boil as us job market simmers. *Bloomberg. Com*, 2018.

[239] How many Low-Code/No-Code platforms are out there? Available: https://www.spreadsheetweb.com/how-many-low-code-no-code-platforms-are-out-there/. [Online; accessed 5-August-2022].

[240] TransmogrifAI - AutoML library for building modular, reusable system. Available: https://transmogrif.ai/. [Online; accessed 5-January-2022].

[241] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web?(nier track). In *Proceedings of the 33rd international conference on software engineering*, pages 804–807, 2011.

[242] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss, and R. Farivar. Towards automated machine learning: Evaluation and comparison of automl approaches and tools. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*, pages 1471–1479. IEEE, 2019.

[243] G. Uddin, O. Baysal, L. Guerroj, and F. Khomh. Understanding how and why developers seek and analyze api related opinions. *IEEE Transactions on Software Engineering*, page 40, 2019.

[244] G. Uddin and F. Khomh. Automatic summarization of api reviews. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 159–170. IEEE, 2017.

[245] G. Uddin and F. Khomh. Automatic summarization of API reviews. In *Proc. 32nd IEEE/ACM International Conference on Automated Software Engineering*, page 12, 2017.

[246] G. Uddin and F. Khomh. Opiner: A search and summarization engine for API reviews. In *Proc. 32nd IEEE/ACM International Conference on Automated Software Engineering*, page 6, 2017.

[247] G. Uddin and F. Khomh. Automatic opinion mining from API reviews from stack overflow. *IEEE Transactions on Software Engineering*, page 35, 2019.

[248] G. Uddin, F. Khomh, and C. K. Roy. Towards crowd-sourced api documentation. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 310–311. IEEE, 2019.

[249] G. Uddin, F. Khomh, and C. K. Roy. Automatic api usage scenario documentation from technical q&a sites. *ACM Transactions on Software Engineering and Methodology*, page 43, 2020.

[250] G. Uddin, F. Khomh, and C. K. Roy. Automatic mining of api usage scenarios from stack overflow. *Information and Software Technology (IST)*, page 16, 2020.

[251] G. Uddin, F. Khomh, and C. K. Roy. Automatic api usage scenario documentation from technical q&a sites. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(3):1–45, 2021.

[252] G. Uddin and M. P. Robillard. How api documentation fails. *Ieee software*, 32(4):68–75, 2015.

[253] G. Uddin and M. P. Robillard. How api documentation fails. *IEEE Softawre*, 32(4):76–83, 2015.

[254] G. Uddin, F. Sabir, Y.-G. Guéhéneuc, O. Alam, and F. Khomh. An empirical study of iot topics in iot developer discussions on stack overflow. *Empirical Software Engineering*, 26, 11 2021.

[255] E. J. Umble, R. R. Haft, and M. M. Umble. Enterprise resource planning: Implementation procedures and critical success factors. *European journal of operational research*, 146(2):241–257, 2003.

[256] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 803–812, 2014.

[257] A. Van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.

[258] P. Vincent, K. Iijima, M. Driver, J. Wong, and Y. Natis. Magic quadrant for enterprise low-code application platforms. *Gartner report*, 2019.

[259] Vinyl platform overview. Available: `https://zudy.com/`. [Online; accessed 5-January-2021].

[260] Z. Wan, X. Xia, and A. E. Hassan. What is discussed about blockchain? a case study on the use of balanced lda and the reference architecture of a domain to capture online discussions about blockchain platforms across the stack exchange communities. *IEEE Transactions on Software Engineering*, 2019.

[261] D. Wang, J. D. Weisz, M. Muller, P. Ram, W. Geyer, C. Dugan, Y. Tausczik, H. Samulowitz, and A. Gray. Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, 2019.

[262] M. Wardat, B. D. Cruz, W. Le, and H. Rajan. Deepdiagnosis: automatically diagnosing faults and recommending actionable fixes in deep learning programs. In *Proceedings of the 44th International Conference on Software Engineering*, pages 561–572, 2022.

[263] R. Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52:376–381, 01 2019.

[264] J. Wexler. Facets: An open source visualization tool for machine learning training data. *Google Open Source Blog*, 2017.

[265] D. Wolber. App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 601–606, 2011.

[266] J. Wong, M. Driver, and P. Vincent. Low-code development technologies evaluation guide, 2019.

[267] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics*, 24(1):1–12, 2017.

[268] M. Woo. The rise of no/low code software development—no experience needed? *Engineering (Beijing, China)*, 6(9):960, 2020.

[269] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, and L. He. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems*, 2022.

[270] Infinable platform documentation. Available: `https://docs.infinable.io/platform/infinable-quickstart/what-can-i-do-with-infinable`. [Online; accessed 5-January-2022].

[271] D. Xin, E. Y. Wu, D. J.-L. Lee, N. Salehi, and A. Parameswaran. Whither automl? understanding the role of automation in machine learning workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2021.

[272] Xojo platform overview. Available: `https://www.xojo.com/products/mobile.php`. [Online; accessed 5-January-2022].

[273] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu. Explainable ai: A brief survey on history, research areas, approaches and challenges. In *CCF international conference on natural language processing and Chinese computing*, pages 563–574. Springer, 2019.

[274] A. Yang, P. M. Esperança, and F. M. Carlucci. Nas evaluation is frustratingly hard. *arXiv preprint arXiv:1912.12522*, 2019.

[275] D. Yang, A. Hussain, and C. V. Lopes. From query to usable code: an analysis of stack overflow code snippets. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 391–401. IEEE, 2016.

[276] Q. Yang, J. Suh, N.-C. Chen, and G. Ramos. Grounding interactive machine learning tool design in how non-experts actually build models. In *Proceedings of the 2018 designing interactive systems conference*, pages 573–584, 2018.

[277] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, 2016.

[278] Y. Zha and S. Pham. Monitoring downhole drilling vibrations using surface data through deep learning. In *SEG Technical Program Expanded Abstracts 2018*, pages 2101–2105. Society of Exploration Geophysicists, 2018.

[279] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang. An empirical study on program failures of deep learning jobs. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1159–1170. IEEE, 2020.

[280] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang. An empirical study on tensorflow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 129–140, 2018.

[281] L. Zhu, L. Bass, and G. Champlin-Scharff. Devops and its practices. *IEEE Software*, 33(3):32–34, 2016.

[282] W. Zhuang, X. Gan, Y. Wen, and S. Zhang. Easyfl: A low-code federated learning platform for dummies. *arXiv preprint arXiv:2105.07603*, 2021.

[283] Zoho Creator platform overview. Available: `https://www.zoho.com/creator/`. [Online; accessed 5-January-2021].