11-9-2022

Serverless Enabled Framework for Machine Learning Applications: Architecture, Solutions, and Evaluations

Boyuan Guan
Florida International University, bguan003@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SERVERLESS ENABLED FRAMEWORK FOR MACHINE LEARNING APPLICATIONS: ARCHITECTURE, SOLUTIONS, AND EVALUATIONS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Boyuan Guan

2022

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Boyuan Guan, and entitled Serverless Enabled Framework for Machine Learning Applications: Architecture, Solutions, and Evaluations, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

S. S. Iyengar

Mark. A. Finlayson

Wensong Wu

Liting Hu, Major Professor

Date of Defense: November 9, 2022

The dissertation of Boyuan Guan is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2022

DEDICATION

I dedicate this dissertation to my dad, Feng Guan, who encouraged me to pursue this degree, and to my mother and wife, for their unconditional love, support, and understanding during this process. I also have a special appreciation to my daughter for the support and deep apology for the time I missed during the years she grew up.

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my advisor, Dr. Liting Hu, for her unlimited support and guidance during my Ph.D. program. Dr. Hu guided me into new research areas and taught me a lot of research experience. She encouraged me to create novel ideas, guided me in preparing research draft, and instantly introduced cutting-edge topics to me. She always encouraged me to attend the top-tier conferences and supported me to take the opportunity to communicate with other researchers and scientists. Her unlimited support helps me to be succeed in my Ph.D. career and create many fantastic research works. She guided me to join the cycle of systems research in computer science and trained me to be a young scientist.

Second, I would like to thank my committees and professors at Florida International University. Many thanks to my committees, Dr. S. S. Iyengar, Dr. Mark A. Finlayson, and Dr. Wensong Wu, for your great encouragement and support during my research work and guided me to deeply explore unsolved problems.

Third, I would like to express my thanks to my Supervisor, the dean of the FIU library, Dr. Jennifer Fu. Thank you for your long-time guidance and support, which gave me a flexible environment to conduct my frontier research across the industry and academia.

Forth, I would like to thank my friends at FIU and Miami, who give me warm supports and brought colorful life during this special journey. Many thanks to Hailu Xu and Pinchao Liu, my teammates in Elves Research Lab. Many thanks to Yekun Xu, Wei Ren, Yuyang Zhou, Tianyi Wang, and to all my friends that I encountered in this beautiful city. Thank you all for being special parts during this journey.

Finally, from the bottom of my heart, I would like to express my gratitude to my parents and family. My father, Feng Guan, who supported me to pursue a higher degree and always encouraged and trusted me. My mother, Jun Fu, who gives me priceless and infinite love and support, which helps me to be here. My wife, Yang Zhang, and My

Daughter, Kaylee Guan who gives her understanding to let me finish my degree. Their love overwhelms everything I have and is so invaluable that it exceeds anything in the world.

ABSTRACT OF THE DISSERTATION

SERVERLESS ENABLED FRAMEWORK FOR MACHINE LEARNING APPLICATIONS: ARCHITECTURE, SOLUTIONS, AND EVALUATIONS

by

Boyuan Guan

Florida International University, 2022

Miami, Florida

Professor Liting Hu, Major Professor

Over the past decade, the popularity of machine learning applications such as recommendation systems, image recognition, real-time alerts, event detection, natural language processing, and online streaming analytics has increased dramatically. However, there is a long-tail problem has been identified for the ML application. The majority of ML applications are concentrated in high-tech and high-profit areas while ML applications are still difficult to reach for local and low-profit businesses. The two factors that cause this slow growth are the domain problem barrier and the rigid infrastructure environment. Since ML application development is different than the traditional software application which consists of data steam feeding, data prepossessing, model training, evaluation, and service update. Unlike traditional software application architecture which the data layer is separate from the business layers, the data is involved in all the steps in the ML application development process. In other words, domain knowledge is required for the overall life cycle of ML application development. This requirement causes big overheads in the workflow and thus, produces the barrier to the widespread of ML applications in local low-profit businesses. On the other hand, the various types of ML applications and the heterogeneous infrastructure environment make the ML application beyond the reach of these local low-profit businesses. Compared to high-tech and high-profit companies, local low-profit businesses can not afford an IT team to implement their ML applications. Pure

cloud solution normally provides template-like services which are hard to adapt to their own business workflow. Even if they are able to implement some low-complexity ML applications within an on-premise private cloud environment, the application performance is hard to meet a production level stability.

In this dissertation, we present two frameworks: dpSmart and StraightLine aimed at solving the domain-specific problem and heterogeneous infrastructure problem for the ML application. dpSmart is a conceptual framework that provides an additional abstraction on top of the existing ML application development process. The purpose of the additional abstraction is to limit the domain-specific knowledge involvement to one single step so that the rest of the steps can be standardized. Straightline is a from-development-to-deployment multiple resources-ware machine learning application pipelines. It first separates the ML application development and deployment phase so that various ML applications can be served without affecting the deployment environment. StaightLine adapts the docker container to provide flexibility for dynamic implementation among heterogeneous infrastructure environments. StraightLine also presents a placement algorithm to maximize the ML application performance.

TABLE OF CONTENTS

CHAPTER PAGE

1. INTRODUCTION 2
1.1 Motivation 2
1.1.1 Domain Specific Problem Challenges 3
1.1.2 Various Applications and Heterogeneous Environments Challenges 5
1.2 Proposed Methodology 6
1.2.1 Noverty 7
1.2.2 Comparison to state-of-the-art 10
1.2.3 Contributions 12
1.3 Summary and Roadmap 13

2. Background 14
2.1 Domain Specific Problem for ML Application 14
2.2 Implementation for ML Application 15
2.2.1 On-premises ML Applications 16
2.2.2 Public Cloud ML Applicaitons 16
2.2.3 Hybrid ML Applications 17
2.2.4 Serverless as additional resource 17

3. dpSmart in Action - a recommendation system for digital library domain . . . 21
3.1 dpSmart: Recommendation System 21
3.2 Introduction 22
3.3 Related Work 24
3.3.1 Recommendation System in Digital Libraries 24
3.3.2 User-group clustering for User Reorganization 25
3.3.3 Stereotyping Recommendation 26
3.4 System Design 27
3.4.1 Automatic Web Server Log Mining Module 28
3.4.2 User-group clustering Module 30
3.4.3 Recommendation Strategy Module 31
3.4.4 Customized Recommenders Module 33
3.5 Evaluations 40
3.5.1 Experiment Setup and Data Collection 41
3.5.2 Multi-process Programming Evaluation 42
3.5.3 Impact to the Hosting Digital Repository Systems 42
3.6 Conclusion 45

4. StraightLine: From-development-to-deployment Multiple Resources-aware Machine Learning Application Pipeline 46
4.1 Introduction 46
4.2 Background and Literature Review 49

4.2.1 Background . . . 49
4.2.2 Existing ML Application Pipelines . . . 50
4.3 Design . . . 52
4.3.1 Overview . . . 52
4.3.2 Model Development Abstraction . . . 53
4.3.3 Multiple Implementation Deployment . . . 54
4.3.4 Real-time Resource Placement . . . 55
4.4 Evaluation . . . 56
4.4.1 Setup . . . 58
4.4.2 Model Development . . . 60
4.4.3 Model Deployment . . . 62
4.5 Conclusion and Future Work . . . 63

5. CONCLUSION AND FUTURE WORK . . . 73
5.1 Conclusion . . . 73
5.2 Lessons Learned . . . 74
5.3 Broader Impact . . . 75

BIBLIOGRAPHY . . . 76

VITA . . . 87

LIST OF TABLES

TABLE PAGE

4.1 Summary of the computational resources for the experiment setup 60

4.2 The official parameters of Xception [Cho17] 60

LIST OF FIGURES

FIGURE PAGE

1.1 A long-tail problem has been identified in the applied field of ML applications. Even though the number of the ML applications are increased dramatically over the past years, they are concentrated in high-tech and high-profit areas like online ADs and Web Searches areas. We identified the challenges of this long-tail problem including Domain Problem Abstraction, Resource Provisioning, Flexibility in Hybrid Infrastructure, and ML Applications Performance Optimization. 3

1.2 The overall contribution of this dissertation is based on the two existing challenges discussed in the previous section which consists of two parts: a generalized methodology for domain-specific problem abstraction and the layered framework for Ml applications implementation in heterogeneous environments. 6

1.3 A conceptual mapping from the standard KDDM process to a Domain-Specific ML Application Development Process. The process is defined as Input Vectors Generation, Domain Knowledge Integration, Model Development, and Model Development. 8

1.4 The ML application implementation challenge is caused by the various types of ML applications, the heterogeneous infrastructure, and the placement problem. 9

2.1 End-to-end machine learning workflow which consists of big data streaming process, data processing, model training, model turning, and service updating. 15

3.1 Overall framework of `dpSmart`. `dpSmart` consists of four components: the automatic web server log mining module, the user-group clustering module, the recommendation strategy module, and the customized recommenders module. The automatic web server log mining module first processes the log data into user vectors. Secondly, the user-group clustering module runs the subspace clustering and maps the clusters into user groups by using dominant features. Thirdly, the recommendation strategy module defines the group preference against different recommenders for the stereotyping filter. Finally, the customized recommenders module generates the final recommendations from specific recommenders. 26

3.2 Sample Decision Tree to demonstrate how to identify the user groups by using the dominant features. Three dominant features, Metadata, Search Terms, and ID-involved are used for user group recognition. The green box demonstrates the recommenders' selection for the different user groups. 33

3.3 The running time latency, the CPU usage for 4,000 sample records, and the memory usage for 4,000 sample records. 37

3.4 The Average Usage of CPU, Memory and Virtual Memory by using 1-process, 3-processes, 5-processes, and 7-processes for the task of running 4000 records. 38

3.5 The Page View Stats for the year from 2015 to 2018, from January to March in year 2018, and from January to March in year 2019. 39

3.6 The Bounce & Drop-off Rate for the year from 2015 to 2018, from January to March in the year 2018, and from January to March in the year 2019. . 40

3.7 The system usability statistics from 2015 to 2018. 44

3.8 The system usability statistics from Jan. to Mar., 2019. 44

4.1 The conceptual life cycle of ML applications. In model development, the streaming data serves as the training data and goes over predefined data pipelines, and ML models are trained and verified with processed data using local infrastructure (e.g., in-house data center, local server, or docker containers). In model development, the trained ML models are then deployed on the infrastructure composed of different computing resources. Then, applications services send requests to the infrastructure through, for example, RESTful APIs. 65

4.2 The workflow and three layers in StraightLine. In Layer 1, StraightLine uses NVIDIA-Docker to define the execution in model development. In Layer 2, StraightLine deploys multiple implementations for ML applications. In Layer 3, StraightLine runs the online resource placement algorithm to place computing resources for upcoming ML requests. 66

4.3 The core information in the dockerfile. It only copies over the necessary resources and the script to start up in the target environment. There is no operating system-related information carried out and no preset provisioning resources are required. 67

4.4 The sample training data set after pre-processing. 68

4.5 The training log demonstrate the significant improvement from the GPU cluster to the model training task. Each step the GPU cluster perform on average 20 times faster than CPU cluster 69

4.6 Training and validation Accuracy and Loss for the same task running by the CPU and GPU cluster respectively. 70

4.7 The confusion matrix generated by the model trained with the CPU cluster and the GPU cluster respectively. The result does not indicate obvious difference between two matrix. 70

4.8 The experiment is designed for applying same batch of load test within 180 seconds with the same input image to different ML application implementations 71

4.9 The performance of a Flask API server implemented on a local web server and in-house data center. **(a)** shows the failure rate of the local web server and the in-house data center. **(b)** and **(c)** show the session length of the local web server and the in-house data center, respectively. Total sessions start from 10 requests per 180 seconds to 2,000 requests per 180 seconds against the Xception model. 71

4.10 The graph shows the comparison for the same Xception application docker implemented on AWS Lambda with 2GB and 3GB provisioned respectively. **(a)** shows the failed rate directly impacted by the provisioned memory. **(b)** and **(c)** indicates the the application latency does not affect much by the load changes. 72

4.11 Results shows serverless implementation is dominating for the high frequency request environment. It significantly enhance the application capacity from 7-8 per second to 30 per second. 72

4.12 This figure shows the Flask API still out perform the rest implementation in terms of response time. 72

ML short = ML, long = machine learning GPU short = GPU, long = graphics processing unit, short-plural-form = GPUs, long-plural-form = graphics processing units,

CHAPTER 1

INTRODUCTION

In this chapter, we first describe the motivations behind our work. Then, we define the problems we are going to address in this thesis and also the contribution we present to solve the problems. Finally, we outline the road map of this dissertation, which we describe in detail in the later chapters.

1.1 Motivation

Machine Learning (ML) has recently evolved from a small field of academic research to an applied field. According to McKinsey's global survey, ML application is largely deployed in standard business processes and applications with nearly 25 percent year-over-year growth [CCH19, KGS+21, PTAJ21, ZWH+21]. For Industry 4.0, devices and machines across factories work on ML-based anomaly detection, edge robotics, or other industry-related. For automatic vehicles, numerous ML-based applications, such as object tracking, object detection, driving decision, and other autonomous driving-related applications emerge [Ins21]. However, even though the ML applications have been presented in various industry fields and many of them are mature in production, Andrew Ng demonstrates a long tail problem for the existing ML applications distribution in a recent TED Talk [AIA]. The long tail problem as shown in Figure 1.1 indicates that the fast-grown ML applications are concentrated in high-tech and high-profit industry fields like online Ads and web searches. On the long tail end, the local brick-and-mortar businesses like the t-shirt maker, or pizza stores showed a slow movement in the ML application implementation even though they needed ML applications to enhance their business model as much as the high-profit businesses. How to empower any business with ML applications

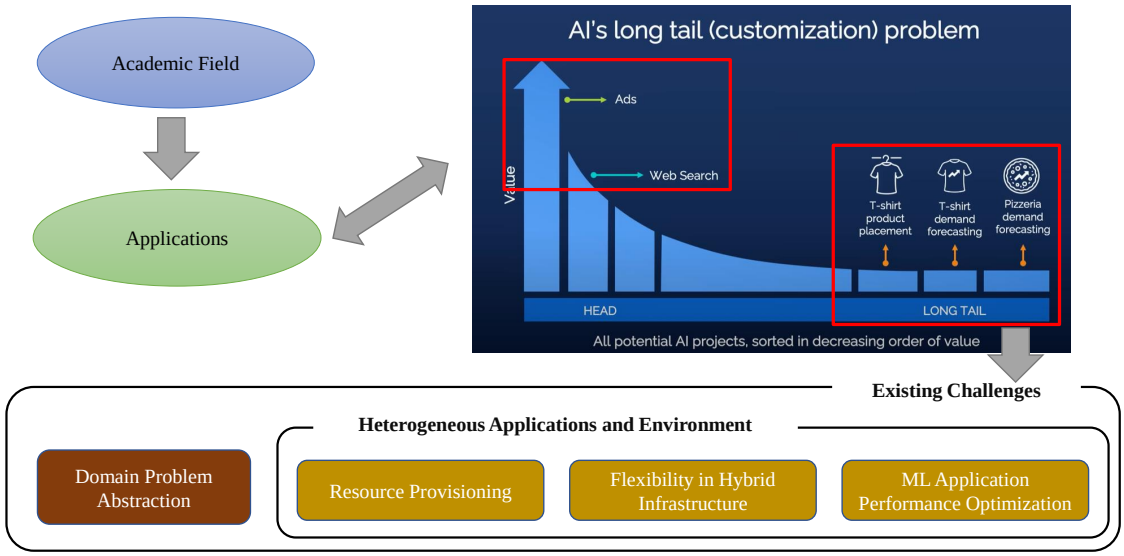


Figure 1.1: A long-tail problem has been identified in the applied field of ML applications. Even though the number of the ML applications are increased dramatically over the past years, they are concentrated in high-tech and high-profit areas like online ADs and Web Searches areas. We identified the challenges of this long-tail problem including Domain Problem Abstraction, Resource Provisioning, Flexibility in Hybrid Infrastructure, and ML Applications Performance Optimization.

and provide democratized access become a big dag between the academic research field and the applied field.

In order to solve or at least remedy this challenge, we first closely analyzed the causes of this problem. We identified two root causes that contribute to this phenomenon, the domain-specific problem challenges and the heterogeneous applications and the environmental challenges. The first one is a business model abstraction problem and the second is a system design problem. In the following sections, we will elaborate on these two challenges.

1.1.1 Domain Specific Problem Challenges

In real-life software system design and development processes, the domain-specific problem is normally considered as a critical challenge for the success of the overall system.

Domain-Specific Language (DSL) [MHS05] and Domain-specific modeling [KT00] are proposed to solve the domain-specific problems in the traditional software design and development process. However, the existing methods are not compatible with the ML application development process. For example, if we need to design a sales system for a bookstore, we need to define the data schema (e.g. book name, publisher, price, quantities, etc.) and develop the function for query, order, and inventory management on top of the data schema. When we need to develop a similar system for t-shirt sales, as long as the data schema is modified to match with the new domain (e.g. t-shit name, brand, price, quantities, etc.), the existing functionality should be carried on from the book sales system and functioning. However, considering a similar situation in ML application development for a book sales recommendation system and a t-shirt sales recommendation system, even though it seems a similar comparison to the traditional software system, the changes between the two systems are much more complicated. The vector, which is used as the input for the model training, built for book recommendation can be the content of the different books like the title, author, publisher, etc. but also can be from the readers like the age, education, genders, etc. On the other hand, the vector build for the t-short recommendations system can be the colors, sizes, materials, or the locations, weather, and cultures. Two types of vectors can be various in both dimensions and the content values and thus, the traditional data schemes method would not work. The model selection can also be different depending on the different vectors. For example, Naive Bayes [WKM10] can solve some tree-based statistical problems but Supported Vectors Machine (SVM) is normally outperformed when for high dimensional conditions. The size of the training data also makes a difference where deep learning is proved to be a better choice when dealing with large training data sets.

The root cause that the domain-specific problem generates more complexity in ML applications than the traditional software application is the difference between the devel-

opment procedure. In the traditional software development process, the data layer can be abstracted independently while the ML application development process, as derived from the data mining process, is highly dependent on almost all the steps. Therefore, an ML application development framework that takes the domain-specific problem into consideration is highly demanded.

1.1.2 Various Applications and Heterogeneous Environments Challenges

Another challenge that prevents ML applications to implement in small-scale businesses is the various application types as well as the heterogeneous deployment environment of the ML applications. For high-tech businesses, normally deal with a single type a single group of similar ML applications which tie to their core business model while small businesses **tend to use different types of ML applications as supporting tools** for their business. For example, a business running an online ADs platform can host a whole IT and data science team dedicated to supporting the Continuous Integration (CI) and Continuous Delivery (CD) [Bat19] of their online ADs model while a local pizza store can benefit from recommendation system for online ordering [BCFPR21], pattern detection for inventory management [dSdAG17], and computer vision for virtual customer services [VKP22]. The heterogeneous deployment environment is the second main difference between ML application usage for high-tech businesses and small-scale businesses. Since small-scale businesses **tend to use multiple types of Ml applications on a flexible and scattered pattern**, they do not require, and normally cannot afford, an IT or data science team to support their ML application solely but still expect these applications to be functioning when the requests come. From the infrastructure perspective, small-scale businesses normally host a mixed type of IT infrastructure including local computational

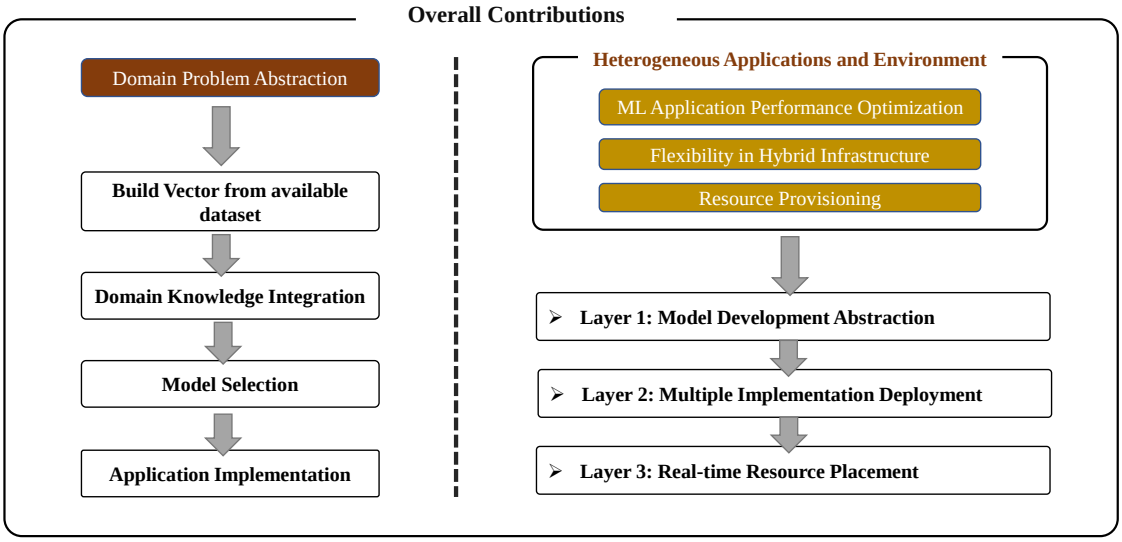


Figure 1.2: The overall contribution of this dissertation is based on the two existing challenges discussed in the previous section which consists of two parts: a generalized methodology for domain-specific problem abstraction and the layered framework for Ml applications implementation in heterogeneous environments.

resources, in-premise private cloud, and public cloud to minimize the operating cost. Therefore, in order to solve these problems, we want to identify a paradigm that can facilitate both the various ML application types and the heterogeneous implementation environment to provide democratized access to small-scale businesses.

1.2 Proposed Methodology

As shown in 1.2, our contributions are split into two categories based on the two existing problems identified in the last section. For the domain-specific problem, we proposed a general methodology to provide a standard procedure for ML application development. For the heterogeneous applications and environments problems, we proposed a serveless-ready framework to allow various types of ML applications to be implement into heterogeneous infrastructure environments. We will discuss the novelty, comparison to state-of-the-art, and our contribution to each category in the following subsections.

1.2.1 Noverty

dpSmart - a generalized paradigm for domain-specific problem abstraction

In order to remedy the domain-specific program challenge, we looked into the existing ML application development procedure to clarify how domain knowledge is involved. According to the knowledge discovery and data mining (KDDM) process, [SOB10], the standard ML application development process consists of the business understanding phase, data understanding phase, the data preparation phase, model phase, evaluation phase, and model deploy phase. As we mentioned in the previous section, the main challenge for the domain-specific challenge is caused that domain knowledge is required in multiple steps in the traditional KDDM process. The practical problem caused by this is that domain experts are needed in all these steps which causes a critical overhead in the real-world workflow. For example, different technical teams like the database engineer team, data analyst team, and data scientists team are working on Data Understanding Phase, Data Preparation Phase, and Model Phase respectively while domain experts are needed in all these three phases. There suppose to be minimal communication required other than the technical documents between teams. However, involving domain experts in these phases requires more communication overheads (e.g. meetings, additional documents, redesigns, and debugs). It also brings more chances to generate deficient designs due to knowledge barriers between the domain experts and the technical teams. Therefore, in the proposed paradigm as shown in Figure 1.3, we introduced a separate step for Domain Knowledge Integration after the Input Vectors Generation and before the Model Development. The benefit of having this abstraction process on top of the traditional KDDM process is that domain experts are only needed in the Domain Knowledge Integration step and thus maximize the efficiency and minimize the communication overhead and chance to generate biased designs.

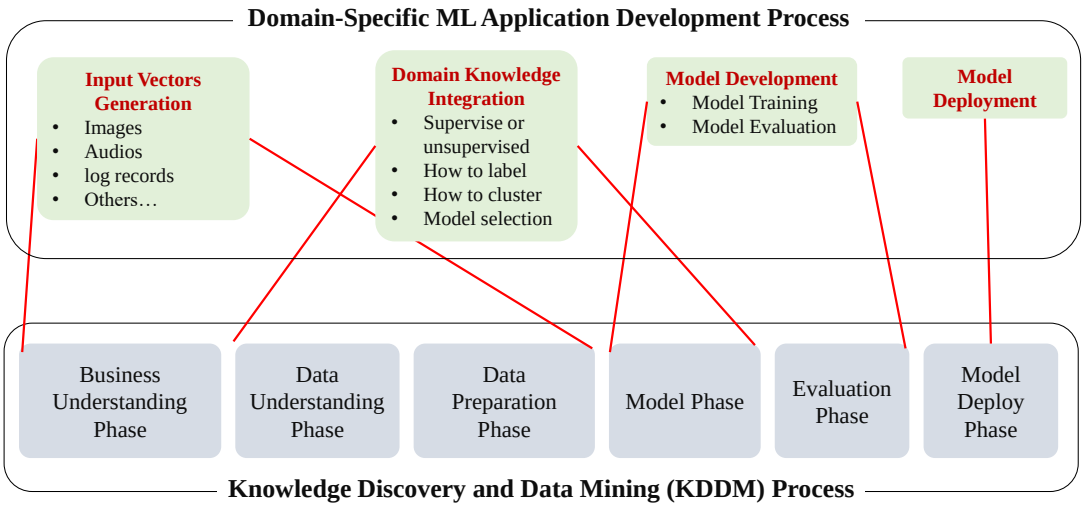


Figure 1.3: A conceptual mapping from the standard KDDM process to a Domain-Specific ML Application Development Process. The process is defined as Input Vectors Generation, Domain Knowledge Integration, Model Development, and Model Development.

To the best of our knowledge, dpSmart is the first framework designed to solve the domain-specific problem for ML application development. It focuses on limiting the domain knowledge involvement into one single step so that the amount of overhead work from the domain experts can be minimized. On the other hand, by removing the domain knowledge dependency, the rest of the steps like vector generation, model development, and model deployment can be standardized and thus, improve productivity.

StraightLine - a serverless enabled hybrid framework

The second problem we identified in this dissertation is the Ml application implementation challenge. As shown in Figure 1.4, there are three questions that need to be answered in order to solve the challenge: *how to provide appropriate resources for different stages or phases?*, *how to satisfy heterogeneous ML applications?*, and *how to adapt to the hybrid infrastructure?*. In this dissertation, we present StraightLine, a from-development-to-

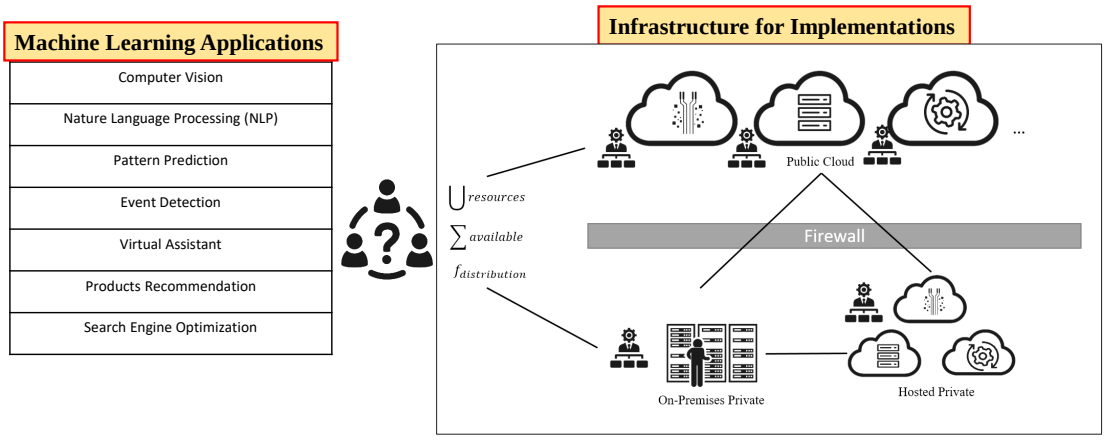


Figure 1.4: The ML application implementation challenge is caused by the various types of ML applications, the heterogeneous infrastructure, and the placement problem.

deployment multiple resources-aware ML pipeline, to address the questions listed above. The key innovation is that StraightLine incorporates the entire life cycle of ML applications and multiple computing resources into one pipeline.

The StraightLine is a from-development-to-deployment ML application pipeline that enables cross-platform ML applications and hybrid infrastructure implementations. StraightLine leverages docker to containerize the stages in model development to offer plug-and-go provisioning, and build up multiple computing platforms and cross-platform containers for model deployment. The detailed contribution is listed below.

To the best of our knowledge, StraightLine is the first framework that covers the ML application life cycle, considers the heterogeneous infrastructure environment, and provides dynamic placement. **StraightLine provides two separate infrastructure environments** for ML application development and deployment so that various types ML applications can be developed in their own environment while deployed in a uniform environment for service deployment. **StraightLline considers heterogeneous infrastructures as a pool of resources for ML application implementation**. The ML ap-

plications in StraightLine can be flexibly implemented in available resources like local servers, on-premise cloud, public cloud, and serverless computing. **StraightLine provides a dynamic placement** for the application requests based on the condition of the real-time resources.

1.2.2 Comparison to state-of-the-art

dpSmart - a generalized paradigm for domain-specific problem abstraction

dpSmart is designed to provide a practical framework for the overall development life cycle of the ML application. The recent research mainly focuses on Domain-Specific Language (DSL) and domain-specific modeling for the ML application. However, most of these research [PDGQ05, CJL+08, ORS+08, WCR+11] are either aimed at providing an interpretation of the data processing steps or solely focus on the model training step. On the other hand, other types of research [TK16, Hes09] are purely focused on the domain-specific business model which overlooks the actual ML application development process.

StraightLine - a serverless enabled hybrid framework

The recent research on pipelines and framework are highly focused on the spot of the ML application life cycle. Elshawi et al. presented a framework that includes the basic clouding computing resource (e.g., storage, CPU, and memory), data streaming platforms (e.g., Hadoop [had], Spark [ZCD+12], Flink [CKE+15]) and workflow environment (e.g., Spark [ZCD+12], Google ML [goo], TensorFlow [ABC+16], KeyStone ML [key]). The proposed framework supports the ML application workflow from data management to model training while it does not support ML application services [ESTT18].

Wang et al. presented Rafiki, an ML analytics service system to provide the training and inference service of ML models. However, Rafiki is limited to a 3-node cluster, so it is not applicable to hybrid infrastructure [WWG+18].

Sánchez-Artigas et al. proposed a Serverless enabled ML system, MLLess, which claims 15 times faster than serverful ML applications and successfully reduces the cost. The paper compared the ML model training process in detail by using PyTorch and PyWren on local data center and serverless environments respectively [SAS21]. However, the experiment is based on the assumption that all the training environments use CPU clusters since the GPU cluster is not supported by Lambda. This generates a critical bias from the real-world applications since GPU cluster has become a fundamental requirement for ML applications.

Naranjo et al. proposed a serverless gateway for event-driven ML inference in multiple clouds. This paper presented a framework that allows packaging the inference function as a RESTful API on both serverless and on-premise cloud [NRMB21]. However, the paper focuses more on the feasibility of the proposed framework. The performance difference between the on-premise and serverless resources is not discussed. The paper also focuses on the inference function based on a pre-trained model. The model development is overlooked even though an AWS Batch is included in the framework which can provide GPU cluster support for the model training.

Shukla et al. proposed an anomaly detection application for Internet-of-Things (IoT) devices over edge computing networks. The data is collected by IoT sensors and sent to the edge for model training and deployment. For model inference, IoT devices send data to edge for model inference and the edge sends the inference results to the cloud for analytics [SS22]. However, the paper only focuses on a single edge-based environment while ignoring the hybrid infrastructure.

We can conclude that most of the works focus on one single stage or phases in the life cycle of ML applications. Moreover, most of the works assume that the infrastructure for ML applications is unified and monotonous. This serves as the motivation of this paper to offer a from-development-to-deployment system for ML applications.

1.2.3 Contributions

dpSmart - a generalized paradigm for domain-specific problem abstraction

In this dissertation, we conduct a use case study by using the proposed paradigm: dpSmart, which applies a recommendation system in a digital library domain. The detailed contributions for this specific domain problem are summarized below.

- Applying dpSmart paradigm provides a flexible framework that allows the digital library can build its recommendation system purely from log data and metadata.
- Facilitate multiple popular recommenders and implement them into a real-world Digital Repository System.
- Minimize the side effect (noisy recommendation) by applying a customizable group-based recommendation strategy
- The experimental evaluation shows that by applying multi-process programming, the model building time can be significantly reduced.
- The system usage statistics also indicate that during the evaluation time from January 2019 to February 2019, the Page Views have increased compared to 2018, demonstrating the effectiveness of our proposed framework.

StraightLine - a serverless enabled hybrid framework

The detailed contribution of the works in this dissertation related to StraightLine is listed below.

- Straightline adapts a divide-and-conque fashion to supply **two independent computational environments** for model development and model deployment respectively so that multiple models can be developed with maximized performance as well as lowered cost.
- Straightline leverages docker to containerize the stages in model development to offer **plug-and-go provisioning** and build up multiple computing platforms and cross-platform containers for model deployment.
- Straightline takes a hybrid infrastructure which includes the **local server, on-premise private cloud, and serverless computing** as part of the design. The ML application is capable of being placed in all these infrastructures.
- To consider the hybrid infrastructure, Straightline designs an **online resource placement algorithm** to allocate computing resources for ML applications, using request frequencies, request data sizes, and processing time as the input of the algorithm.

1.3 Summary and Roadmap

The rest of this dissertation is organized as follows. We introduce the background details in Chapter 2, then we describe the dpSmart as an instance of the recommendation system for the digital library domain in Chapter 3. We next show the design and details of StraightLine as a from-development-to-deployment multiple resource-aware in Chapter 5. Finally, we conclude this dissertation in Chapter 6.

CHAPTER 2

BACKGROUND

In this section, we will discuss the background for the domain-specific ML applications as well as the implementation environment for the ML applications. For the domain-specific ML application, since there is no specific systemic research, we will go through several recent pieces of research. For the implementation environment for the ML application, we will discuss the model development abstraction, multiple infrastructure deployment, and real-time resource placement respectively.

2.1 Domain Specific Problem for ML Application

Domain-specific ML applications are still in their infancy and are considered as the main obstacles for Ml applications to implement in non-tech industries. In 2011, a Domain-Specific Language (DSL), OpiML [SLB+11], for machine learning was proposed to solve the problem. OpiML derives the DSL concept (e.g. HTML for web browsers) from traditional software development procedures to provide an abstraction layer so that different models can be used for heterogeneous domains. However, as shown in Figure 2.1, the development procedure is the main difference between ML application development and the traditional software development procedure. OpiML failed to provide a practical framework for the overall Ml application development cycle. Several other DSL-based kinds of research [PDGQ05, CJL+08, ORS+08, WCR+11] have been published but failed for the same reason. A recent research [GMGC22] provides a details procedure to design DSL for the dataset. However, it is limited to the data processing step.

There are more research has been published recently for [TK16, Hes09] domain-specific modeling. However, these types of research focus on more on the business model

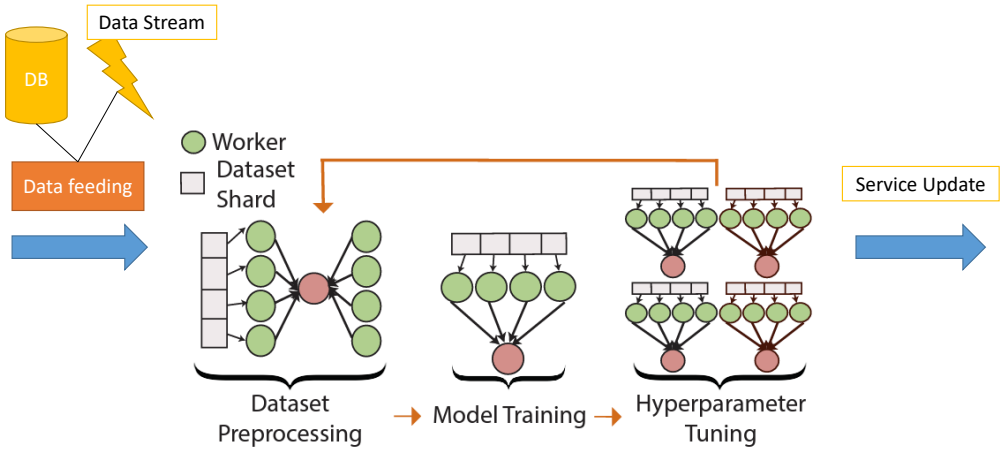


Figure 2.1: End-to-end machine learning workflow which consists of big data streaming process, data processing, model training, model turning, and service updating.

abstraction which is hard for engineers to follow in the ML application development process.

By reviewing recent literature on DSL for ML applications and domain-specific modeling, we can conclude that all this research is either focused on the interpretation of the data processing or the abstract of the model training process. A domain-specific ML application development framework that supports the overall development cycle is in demand.

2.2 Implementation for ML Application

Based on a commercial analysis report [Ins21], it indicates that the mainstream ML applications implementation is still on Cloud and On-premise. It is also an increasingly popular concept for AI as a Service [ES20] which provides AI functions as services to the end users. There are two obvious problems for the ML application lifecycle: **Explicit re-**

source management and **Over-provisioning**. In the following sections, we will discuss how serverless components can address these problems and improve the performance of each type of implementation.

2.2.1 On-premises ML Applications

In an on-premises environment, resources are deployed internally and within an organization's IT infrastructure. The end user is responsible for maintaining the solution and all associated processes. Although the number of public cloud solutions has increased significantly in recent years, almost half of the market share of ML applications is still implemented on-premises, as the public cloud does not offer complete control and a high level of security. The obvious disadvantage of on-premise implementation is the cost of equipment and labor. Organizations that implement software on-premise always suffer from the ongoing costs of server hardware, power consumption, and space. The established approach of freeing up low-level VM resources, such as memory and CPUs, places a significant burden on ML developers who face the challenge of dealing with additional IT tasks in addition to their ML workloads.

2.2.2 Public Cloud ML Applicaitons

Companies that opt for a cloud computing model normally will go with a pay-as-you-go pricing model, without incurring any maintenance and upkeep costs. The price adjusts according to consumption. Considering that ML applications routinely perform a number of different tasks during model training and place heavy demands on computing resources, the cost savings of opting for a public cloud solution is enormous. However, the issue of data ownership is one that many companies - and vendors - struggle with. Data and encryption keys reside with your third-party provider. So if something unexpected hap-

pens and there is an outage, you may not have access to that data. Security concerns also remain the biggest barrier to public cloud computing. There have been many publicly disclosed cloud security. From employee personal data, such as login credentials, to intellectual property loss, the security threats are real.

2.2.3 Hybrid ML Applications

While the debate over an on-premises data center versus a cloud computing environment is a real one that many organizations are currently having in their offices, the hybrid cloud implementation has also caught on because it can flexibly combine public cloud and private cloud resources, maximizing cost savings and productivity while minimizing latency, data protection, and security issues. However, the hybrid cloud offers an either-or solution based on the assumption that the existing application ML must have a decoupled modular design that allows some modules to be allocated in the public cloud and some modules in the on-premises data center.

2.2.4 Serverless as additional resource

In the application layer, a variety of services and applications are supported by a serverless framework in the edge computing environment, such as event data processing, IoT data monitoring, video transcoding, and so forth. By looking at some of the unique features of the applications in the edge computing domain, it is quite obvious that the serverless framework provides a complementary conceptual model to accommodate the needs of edge computing applications which include:

- Providing a layer of abstraction to solve the un-controllable hardware on the edge
- Enabling rigid sandboxing for an untrusted environment

- Providing a scalable and elastic model for processing unpredictable real-time data at scale
- Maintaining the compute efficiency and small memory/storage footprints to operate in a low resource environment to reduce the expense

Serverless has evolved significantly since its inception. Tools, both on the vendor side and in the open-source space, have improved massively. In parallel, practitioners have developed best practices and mental models for working in a serverless environment. All of these advances will enable the rise of the serverless edge application environment. Given the popularity of serverless edge computing, which is still in its infancy, there is still much room for researchers and developers to explore.

We conduct a thorough review of the published paper related to the applications in serverless from the year 2019 to year 2022 to summarize the state-of-the-art applications in this specific framework. The applications are spread out in the domains of IoMT/Health Care, Smart Cities/Smart Farms, Edge/Serverless Machine Learning, Industry 4.0, and 5G networks. Even though the research focus and the motivations are differed based on the nature of the domain specifics, most of them have the machine learning component which indicates that serverless computing is a great development environment for ML applications.

The **Internet of Medical Things (IoMT)** and **Health Care** are one of the most popular domains that apply the serverless and edge computing framework. Since the main challenge for IoMT is the requirement of the near-zero latency application on the edge, the applications and research are mostly focused on this particular problem and providing solutions from different aspects. A survey paper [AFK+22] was published in 2022 on the topic of the utilization of mobile edge computing on the IoMT. Serverless computing is considered a significant booster to successfully implementing the 5G based Mobile Edge Computing (MEC)-based model. There are also several papers that focus on

real-world applications or case studies in IoMT. [BAS+21] and [WWL+21] explored the Browser as Edge method to minimize the latency to implement TinyML machine learning model to help real-time data monitoring and face mask detection. In [NAC+22], a new IoMT framework for real-time electroencephalography (EEG)signal monitoring and analysis using an explainable artificial intelligence (XAI) technique is presented. As real-timeliness is the key aspect of the growing perspective of the Internet of Things (IoT), [RDD19] presents a novel dew computing architecture that provides in-depth analysis and validation of real-time scheduling in resource-constrained hardware platforms through real-time message passing in presence of a cloud facilitator with a real-time context. The case study in this paper was conducted under Lamda, the AWS serverless solution. [APP20] provide a description of the key technologies and paradigms related to healthcare 4.0 and discuss their main application scenarios; they also provide an analysis of (iii) the benefits carried and the new interdisciplinary challenges.

Smart cities / Smart farms are another pioneered area to apply the serverless and edge computing framework. Studies in this domain are focused more on the platform and architecture design because the main challenge is how to seamlessly integrate heterogeneous data sources. Deployed the business solution through Serverless Function as a Service (FaaS) is a key component in this design. A comprehensive survey [WWGV22] of currently known projects and startups on blockchain-based transactive energy for the cross-sector local community with buildings and electric vehicles will be published in 2022. Another paper [IAS22] presents an energy-efficient opportunistic model utilizing the ant-based routing algorithms for LS-WSN SC waste management applications. Smart farms, as a sub-model of the smart cities concept also drawn attention in the past several years. [ASCG+20] presents a platform focused on the application of IoT, edge computing in smart farming environments, designed to monitor the condition of dairy cattle and feed grains in real-time and to ensure the traceability and sustainability of the various processes

involved in production. in [PMD21], an optimized version of the Kappa architecture is presented, enabling fast and efficient data management in agriculture.

Other research is also published in domains like **general data interoperability, edge machine learning, 5G network model, and online multimedia content** from 2019 to 2022. [MLLI21] conducted a case study that focused on data interoperability. The study revealed that by having both FaaS and BaaS, the serverless framework provides a flexible middleware for existing IoT platforms that can significantly mitigate the data interoperability problem. [BTK22] and [CFT+18] proposed a general approach of implementing the machine learning model into an edge computing framework with serverless architecture enabled and a case study for machine learning model implementation in a serverless framework respectively. The performance and effectiveness are evaluated by exploiting a holistic end-to-end image classifier, a famous machine learning use case in the MNIST dataset. The proof of concept provides comprehensive assessments that prove the effectiveness of latency reduction and distributed machine learning deployment. [DWW+22] proposed a stateless design in the 5th generation core (5GC) network to solve the coupling of protocol states and functions problem. Online multimedia content is also a trending topic because of the data-intensive workloads. [SNP19] proposed a solution for collecting, storing, managing, and accessing online media data.

The only research that we can identify for Ml application life cycle deployment is a recent thesis [Tet22] in 2022 which presents a machine learning workflow model and serverless deployment orchestration. However, the method is featured for technology-independent modeling and only uses serverless computing for the implementation method. The heterogeneous infrastructure condition is not considered.

CHAPTER 3

DPSMART IN ACTION - A RECOMMENDATION SYSTEM FOR DIGITAL LIBRARY DOMAIN

We proposed dpSmart as a framework to solve the domain-specific problem for ML applications. dpSmart is designed to limit domain knowledge involvement so that the development procedure can be standardized. We apply this framework to a real-world domain, a digital library, to demonstrate how we can follow the standard data mining procedure and integrate the domain knowledge at the same time.

3.1 dpSmart: Recommendation System

Digital Repository Systems have been used in most modern digital library platforms. Even so, Digital Repository Systems often suffer from problems such as low discoverability, poor usability, and high drop-off visit rates. With these problems, the majority of the content in the digital library platforms may not be exposed to end users, while at the same time, users are desperately looking for something which may not be returned from the platforms. The recommendation systems for digital libraries were proposed to solve these problems. However, most recommendation systems have been implemented by directly adopting one specific type of recommenders like Collaborative-Filtering (CF), Content-Based Filtering (CBF), Stereotyping, or hybrid recommenders. As such, they are either (1) not able to accommodate the variation of the user groups, (2) require too much labor, or (3) require intensive computational complexity.

In this paper, we design and implement a new recommendation system framework for Digital Repository Systems, named `dpSmart`, which allows multiple recommenders to work collaboratively on the same platform. In the proposed system, a user-group-based recommendation strategy is applied to accommodate the requirements of the different

types of users. A user recognition model is built, which can avoid the intensive labor of the stereotyping recommender. We implemented the system prototype as a sub-system of the FIU library site (`http://dpanther.fiu.edu`) and evaluated it on January 2019 and February 2019. During this time, the Page Views have increased from 8,502 to 10,916 and 10,942 to 12,314 respectively, compared to 2018, demonstrating the effectiveness of our proposed system.

A framework for serverless machine learning needs to meet three critical goals. First, its API needs to support a wide range of ML tasks: dataset preprocessing, training, and hyperparameter optimization. In order to ease the transition from existing ML systems, such API should be developed in a high-level language such as Python. Second, to provide storage for intermediate data and message passing between stateless workers, it needs to provide a low-latency scalable data store with a rich interface. Third, to efficiently run on resource-constrained lambdas, its worker runtime needs to be light-weight and high-performance.

3.2 Introduction

Digital Repository Systems have been used in most modern digital library platforms, which are used to store, archive, and index all of the digital assets in the library as well as to serve those assets to clients and users of libraries over the Internet. Digital Repository Systems have three long-existing issues including (1) the low discoverability of the content; (2) lack of assistance to explore the system, and (3) high drop-off visits. `dPanther` [dPa] (`http://dpanther.fiu.edu`) is the Digital Repository Systems developed and implemented by Florida International University (FIU) to host the digital assets including digital archives and digital-born content and publications. The web server log of

dPanther in 2019 shows that only 24.9% of the content has been discovered, 62.1% intended search ended with a page review, and 58.8% resulted in drop-off visits.

In the past decade, many studies have been conducted to remedy these issues. The most famous one is the recommendation systems [GMG01] [BLGN13] [BVdB08] [CGLL08] [GP06]. Recommendation systems, as a subclass of information filtering systems, are typically used to produce a list of recommended content from the hosting system by predicting the ”rating” or ”preference” of the users. They improve content discoverability, provide guidance, and retain more users for the hosting system. However, most recommendation systems have been implemented by directly adopting one specific type of recommender like Collaborative-Filtering (CF) [ERK$^+$11], Content-Based Filtering (CBF) [LDGS11], Stereotyping [LPWG14], or hybrid recommenders [Bur02]. As such, they are either (1) not able to accommodate the variation of the user groups, (2) require too much labor, or (3) require intensive computational complexity.

In this project, we design and implement a new recommendation system framework for Digital Repository Systems, named dpSmart, which allows multiple recommenders to work collaboratively on the same platform, including the Content-Based filtering (CBF) [LDGS11], Collaborative Filtering (CF) [ERK$^+$11], Global Relevance (GR), Query Suggestion (QS)/Term Suggestion (TS) [BGLB16] and Location Based Filtering [LZX$^+$08] models. The users are grouped by a stereotyping model and different users are served by different recommenders. The user vectors are built based on the user behaviors extracted from the log data. By optimizing the algorithm with multi-processes programming fashion and maximizing the server capacity, the model re-training time can be reduced significantly.

We implemented the system prototype as a sub-system of the FIU library site (http://dpanther.fiu.edu) and evaluated it on January 2019 and February 2019. During this

time, The Page Views have increased from 8,502 to 10,916 and from 10,942 to 12,314 respectively, compared to Page Views in 2018.

The technical contributions of this paper are as follows:

- We implement a stereotype-based recommendation system that can adapt to multiple different recommenders.
- The proposed system avoids intensive labor and automates the process from the log data extraction to model training.
- By using a stereotyping recommender, we avoid the need for personally identifiable information user data and we reduce the noise recommendation.
- By implementing multi-process programming, the model re-training time can be significantly reduced.

3.3 Related Work

3.3.1 Recommendation System in Digital Libraries

Due to the uncertainty of the content, the variation of patrons, and the unpredictable user interactions with the library system, content-based recommendation systems have dominated in digital library domain. C.Musto et al. [MNL+10] proposed a Content-Based Recommender System for Digital Library for Cultural Heritage. S.Philip et al. [PSJ14] proposed a Content-Based approach in paper recommendation systems for a digital library. However, the problem of a Content-Based recommender lies in that this type of systems are not able to accommodate the variation of the clients and the user behaviors. Another type of recommendation system is customization and personalization for the niche domain. A.F.Smeaton et al. [SC05] proposed a method to personalize a recommendation system for a digital library platform. However, this type of system is very hard

to generalize. In order to generalize and further potentially automatize the customization and personalization process, the user recolonization [SMKL16] was proposed.

In the Research-Paper Recommendation Systems survey [BGLB16], 200 publications have been reviewed and 62 different methodologies have been proposed. Among those proposed methods, content-based filtering is in the top rank, and 55% of the methods are either based on this or leveraged it. Collaborative Filtering ranked No.2 with 18% of methods used. Graph-based recommender ranked No.3 with in 16% of methods used. Other recommenders like stereotyping, item-centric, and hybrid recommendations are mentioned in 11 of the publications.

3.3.2 User-group clustering for User Reorganization

Based on the designed framework, user modeling and user group recognition are key functions to building a flexible recommendation system. Previous publications regarding user modeling have also been reviewed. A recent dissertation from the University of Cornell [Bee17] has been published to introduce Mind Mapping based user modeling for research paper recommendation systems. Although the result is very promising, the approach heavily relies on an open-source JAVA application for managing PDF files, annotations, and references with mind maps called Decear and required labeled user data. Considering that lack of personally identifiable information, user behavior data is one of the toughest challenges for the recommendation system in Digital Library Systems hosted by public libraries, we switch the machine learning approach to an unsupervised learning method to recognize the user group. A good example can be found in [CZYL14], which proposed a clustering-based method for the web services recommendation system. Although the method proposed in [CZYL14] is in the web services recommendation domain, it provides a solution for the problem of lack of labeled user behavior information.

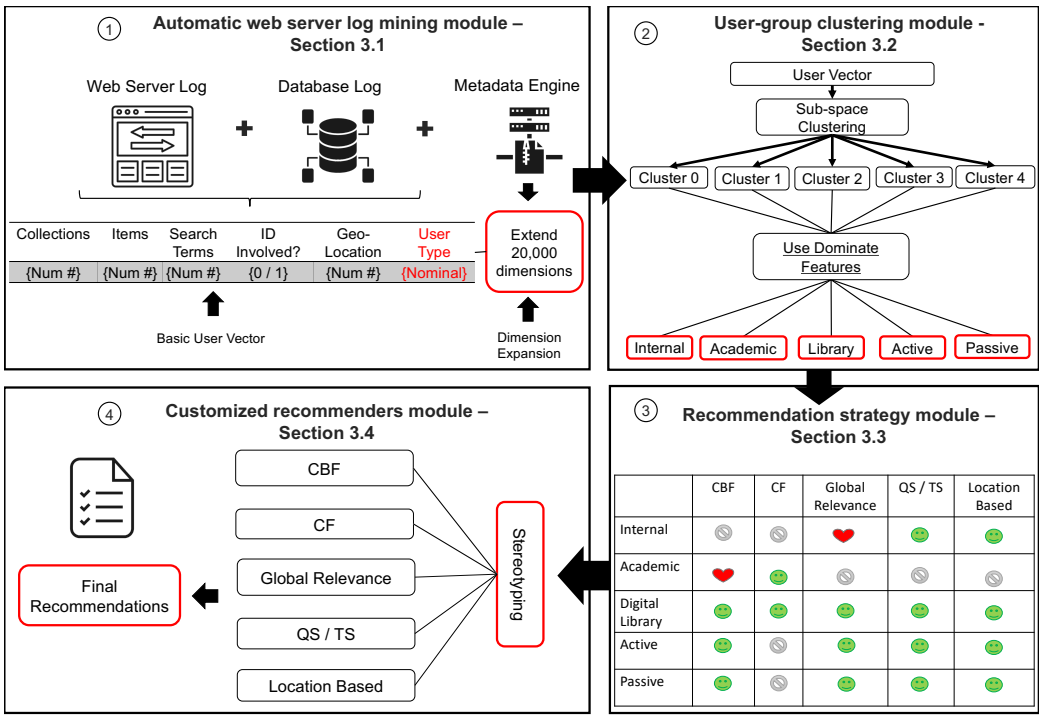


Figure 3.1: Overall framework of `dpSmart`. `dpSmart` consists of four components: the automatic web server log mining module, the user-group clustering module, the recommendation strategy module, and the customized recommenders module. The automatic web server log mining module first processes the log data into user vectors. Secondly, the user-group clustering module runs the subspace clustering and maps the clusters into user groups by using dominant features. Thirdly, the recommendation strategy module defines the group preference against different recommenders for the stereotyping filter. Finally, the customized recommenders module generates the final recommendations from specific recommenders.

A more advanced subspace clustering-based method [NSRK14] was proposed in 2014 for user group identification. This subspace-based clustering method has two main features: prune non-promising features for subspace extension dimensions and fault tolerance.

3.3.3 Stereotyping Recommendation

Stereotyping has a long history for user modeling. Rich [Ric79] in 1979 initially described how to build user models by using stereotypes. In 2007, additional research was

conducted by Guy [Ric79] to implement Stereotyping Recommendation method in media systems. The performance of stereotyping Recommendation, by combining content-based filtering and collaborative filtering, outperforms the regular Adhoc hybrid recommendation system performance. As such, stereotyping recommendation has been considered as the recommender with the best performance. However, the drawbacks of Stereotyping recommenders, such as pigeonholing users and labor intensiveness, have significantly limited the usage of stereotyping recommenders. These two drawbacks are directly caused by the traditional way of manual processing process to identify user groups.

3.4 System Design

We design and implement a new recommendation system framework for Digital Repository Systems, named `dpSmart`, which allows multiple recommenders to work collaboratively on the same platform. `dpSmart` has four major components: 1) the **automatic web server log mining module** to generate user behavior data, 2) the **user-group clustering module** for automatic user group recognition, 3) the **recommendation strategy module** for stereotyping filter based on different user group, and 4) the **customized recommenders module** to implement into `dPanther` Digital Repository Systems.

Figure red3.1 shows the overall design of `dpSmart`. The workflow starts with gathering and processing log data from an IIS web server. The automatic web server log mining module is responsible for processing the data to generate the user vectors. Once the user vectors are generated, the User-group clustering module runs the cluster and generates the group of users. The recommendation strategy module then maps the generated cluster to the corresponding user group and establishes the stereotyping filter. Based on the stereotyping filter, the customized recommender modules can provide customized recom-

mendations from embedded recommenders. The detailed design and implementation of each module are discussed below.

3.4.1 Automatic Web Server Log Mining Module

We utilize two types of resources (i.e., the web server log and the database log) to analyze user behaviors legally. Both types of log data are naturally generated whenever a web-based application is published and very limited user information is stored. We generate datasets through the following data preparation and processing methods, which can be used directly to build user vectors.

The IIS logs are the primary resource for our system because they contain some key information, including but not limited to:

- *The client IP address*
- *The time of the activity*
- *The region of where the user comes from*
- *The type of activities the was interacting with the system*

This information is used as dimensions for building the user vectors. The original IIS log has a standard schema. Note that we assume that each IP on a particular date represents a single user, so a user vector is generated from the combination of IP address and the date from IIS logs, IP-date log.

We first process the raw IIS logs and make them more useful by filtering out logs from bots, local access, scripts, style sheets, and another file loading. We then rebuild the logs by combining the IP and date to create basic user logs that contain the IP activity of the website on a single date. Next, we convert each IP-date log into an IP-date user vector, and each user vector contains all the numeric item numbers they accessed using

the number of transactions. The basic user vector that we can build directly from the IIS log is a vector containing two dimensions which are the number of entries accessed in the database and the number of visits from this IP address.

Secondly, we start expanding the dimensions. Since the IP address contains the location information, after geo-coding the IP address, two additional dimensions (latitude and longitude) can be added to the vector to generate a fourth-dimensional vector. This base vector can provide information about where a particular user came from, how often users access our system, and whether the user has successfully accessed a particular item in the system.

The dimensions can be expanded more by integrating the database query log. The query log also records the contents of the query as well as the IP address. Since the search action is a very strong indicator of the willingness of a particular user to find content in the system, the number of searches made by a particular user is added as an additional dimension of the user vector. In addition to the number of queries, the search content is also very useful. One of the key terms that conveys the clearest information is the system ID of the project in the database. Since the system ID is defined using a specific name convention, it can be identified from the regular expression. Users who use the system ID as a search term are likely to be internal users from our organization or academic users from close partner organizations. Therefore, the system ID is added to the user vector as another dimension. As shown in Figure 3.1 step 1, the generated six-dimensional user vector is used as the basic user vector. However, the six dimensions are not sufficient to build any reliable clustering results. Therefore, we must introduce the metadata of the accessed project to expand the dimension.

Last but not least, the metadata of the record can provide a lot more dimensions. One of the most unique features of a Digital Repository System or digital library system is that all records in the system have prepared metadata that is stored in the metadata engine.

With this feature, we can immediately increase the user vector's dimensions to 50, 100, or more by linking user behavior to metadata. For example, when visiting a particular project, such as the Miami 1920 album, users may also be interested in other historical albums, or they may be interested in other projects by photographer Eva FitzGibbon Drummond, or they may be interested in items in the same collection. In this case, with Coral Gables Memory and Miami Metropolitan Archive, we need to carefully select the metadata tags because we want to collect enough data to fit the cluster to our predefined user groups, and we don't want to introduce too many dimensions. So, after building the basic user vector, we can further expand the dimension by introducing metadata information in the metadata engine.

3.4.2 User-group clustering Module

Since we pre-define the target user group and recommendation strategy, for any active users, as long as we can identify the user group to which the user belongs, we can dynamically provide appropriate recommendations and minimize noise recommendations.

Ideally, if there are fixed metrics that distinguish users in these groups, we can simply identify the user group based on the criteria. However, these user groups overlapped and it is difficult to find a clear boundary between user behavior patterns among groups. as shown in Figure red3.2, we use several dominant features to separate the user group. For example, a very powerful indicator is the appearance of a system ID in a search term. Most likely, the system ID is for internal users only. However, some academic users also save the system-generated ID for quick access to their favorite projects. Another good example is location information. If the location information is internal to the library, then the access should be directed to the internal user group. However, many users of the academic group also work in the library. The identification of the user group can

be achieved by using an indicator from the log data. As shown in Figure 3.2, a sample decision tree illustrates this point. Note that in order to use a tree-based classifier, we have to find the tagged training data, which is not available in existing systems. Therefore, clustering methods are applied in our research.

- *Number of visits*
- *Search conducted*
- *Number of Items*
- *ID involved in search*
- *Location of the users*

In the `dpSmart` framework, a subspace clustering method [NSRK14] is used to generate the user cluster. Based on a survey [KKZ09] published in 2009 for the algorithms of subspace clustering are two major subspace algorithms: bottom-up versus top-down approaches. We choose a topical top-down algorithm, PROCLUS [APW$^+$99] for our framework, since it allows us to define the number of clusters generated for the data set. The software WEKA and a third-party package, Opensubspace v3.31 is used to generate the user cluster.

3.4.3 Recommendation Strategy Module

Similar to other information management systems, Digital Repository Systems have predefined target user groups. The target user groups can be divided into five groups, namely the Library Internal Users, Digital Library Users, Local Academic Users, Active Web Users, and Passive Users. The detailed definitions for each group are listed below:

1. **Library Internal Users** - the users from our internal organization like the developers, the librarians, the system administrators, and metadata creators.

2. **Academic Users** - professional users who are familiar with Digital Repository Systems as well as the content inside the system. This user group mainly consists of scholars/professors, undergraduate/graduate students, and researchers.
3. **Digital Library Users** - the users who are familiar with the structure and interface of Digital Repository Systems. This type of user can retrieve information from the system by following the metadata structure. They are normally not as focused on one specific topic.
4. **Active Web Users** - users who are interested in using the system to conduct the search and try to explore the content but have difficulty finding the right content or are not able to access any content.
5. **Passive Web Users** - users who only "stop by" the landing page without doing anything.

According to this grouping, we further develop a recommendation strategy as shown in Figure 3.1 Step 3. The recommendation preference is marked in different types of attitudes, the red heart if preferred, smile face if they don't bother, and block sign if they dislike. Internal users who are metadata creators, digital asset owners, and system administrators, don't want any type of content-based or collaborative filters. They are keen to Query Suggestions, Term suggestions, and Location-based suggestions. Global Relevance, which makes recommendations based on the system status like the popular items or newly added items, is the preferred recommendation for Internal Users.

Academic users need a completely different recommendation strategy than internal users. These users are very professional and well-trained in Digital Repository Systems or Digital Library Systems, so they really reject irrelevant information. Therefore, global relevance, QS/TS, and Location Based recommender should be blocked for them. A Content-based recommender is the one they prefer and Collaborative filtering is not con-

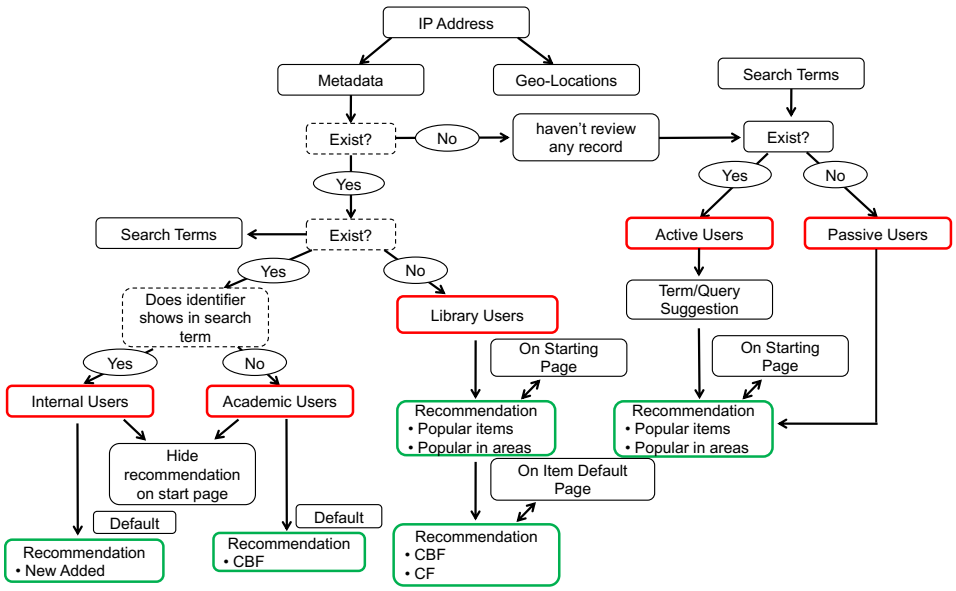


Figure 3.2: Sample Decision Tree to demonstrate how to identify the user groups by using the dominant features. Three dominant features, Metadata, Search Terms, and ID-involved are used for user group recognition. The green box demonstrates the recommenders' selection for the different user groups.

sidered as noise recommendation. Digital Library Users are familiar with digital collection systems but do not have background knowledge of the content in the system, so it is difficult for them to come up with a clear research area. Since these are the main target users, our recommendations are hard to judge for this user group. For these two types of users, we just hide collaborative filtering, since users in the two groups do not necessarily have many similarities in preferred content.

3.4.4 Customized Recommenders Module

As mentioned before, we utilize the Content-based Filtering (CBF), Collaborative Filtering (CF), Global Reference (GR), Query Suggestion (QS)/Term Suggestion (TS), and Location-based Recommendation in the module. These five recommenders are imple-

mented separately and coordinated according to the results of the stereotyping filter. The details of the custom design are shown as below.

Content-based Filtering (CBF)

Based on the record curation process, there are two suitable models to create the item-similarity model: (a) Jaccard [RV96], which is good at depicting similarities between two sets and (b) Directed Graph [JW02], which is good at depicting the similarity of two vectors. Zhou [ZZY+08] presented the challenge of eliminating noise and sparsity using Graph-based method, which is mainly caused by the metadata type and content. The same problem may be more challenging in `dPanther`. A feature of `dPanther` as a Digital Repository System is that it provides a large collection of research articles, research papers, engineer studies/reports, newspapers, government documents, music scores, and many other types of historical documents. From this perspective, the Digital Repository Systems actually provide services for multiple metadata types, which may be completely different. As such, it is very difficult to build directed graphs for records because there are many missing links when moving across different metadata types. Therefore, we choos to establish a similarity model based on Jaccard similarity. Record vectors in the `dPanther` system can be constructed by directly using metadata tags and values. By applying Jaccard similarity, the similarity between items in `dPanther` can be calculated as follows:

$$S_(a,b) = \frac{|A_a \cap B_b|}{|A_a \cup B_b|} \tag{3.1}$$

- Where S is the Jaccard Similarity
- a, b are any two items in `dPanther` system
- A, B are the set of metadata fields and value of item a and b

However, even if this provides default similarity between items in the dPanther repository, the result does not accurately represent the actual relationship between the items, because in the current method, all metadata fields in the collection are equally important, but actually, the metadata fields within the project set have an explicit priority. For example, subject keywords in a set actually play a major role in calculating similarity. If two items share one subject keyword, its impact on the similarity calculation should be greater than the impact of sharing one publisher. Therefore, we modified the original Jaccard method to accept weight data set. Thus, we have:

$$SW_(a,b) = \frac{|\sum_{n=1}^{\infty}((A_n \cap B_n) * W_n)|}{|\sum_{n=1}^{\infty}((A_n \cup B_n) * W_n)|} \tag{3.2}$$

- Where *SW* is the weighted Jaccard Similarity
- *a, b* are any two items in dPanther system
- *A,B* are any pair metadata field from n selection metadata data fields between item a and b
- *w* is the weight assigned for a specific pair of one metadata field

By implementing the weighted Jaccard Similarity, we have to build a reliable measurement system for the items' content. For each individual item, we can have a corresponding ranked similar item list. Then, the next major step is to design another method to generate measurements for items from a user-similarity perspective.

Collaborative Filtering (CF) For public library systems, there are two main obstacles to track user information: (a) The privacy policy limits user information and (b)lacks rating motivation. The process of accessing digital collections is considered part of user privacy (*https://library.fiu.edu/using-the-library/patron-privacy-policy*). Even if the system has a user account function, the users' account activities are not allowed to be used for this type of research. Besides, unlike Amazon or Ebay, the preferences for the items

are very individually based, thus it is hard to find a motivation for the users to provide rating for the digital collections they have viewed in the system. What is more, even if the rating is available, it may not necessarily be useful across the users' groups. Therefore, in the design of the `dpSmart` recommender, the only available resource to use is the weblog which contains the IP addresses of the visits. In our research, we assume one IP address on the same day represents a specific user. From the IP address, we can get an idea of the region where the users came from, the time they visited the system, and the topics they are interested in, and then associate the items with this information.

By considering the information from the user visits, we can take advantage of implicit feedback alternatively. Implicit feedback strongly relates to the behaviors of users. There are two ways to get implicit feedback. One way is based on whether users check on the content of items in history. If so, we assume that users may like items and rate 1 as a score, otherwise rate 0. In contrast, the other way is to count the number of visits for the same item from one specific user. Since one "hit" from the web server log does not necessarily mean one visit from the client side, it may also indicate the user downloading the picture from the item detail page, clicking the link from the client detail page, or playing the video or audio from the detail page. Therefore, this count does not only indicate the actual visits from the user but also provides the interest level of the user.

By considering the two implicit feedbacks, we can get the scores for user-item similarity. The second step is to calculate the similarities between items. Assuming we have two items i and j, if we get scores based on whether users check on contents, then the similarities between i and j can be calculated using standard Cosine Similarity [LHM+14]. The basic Cosine Similarity is define as below:

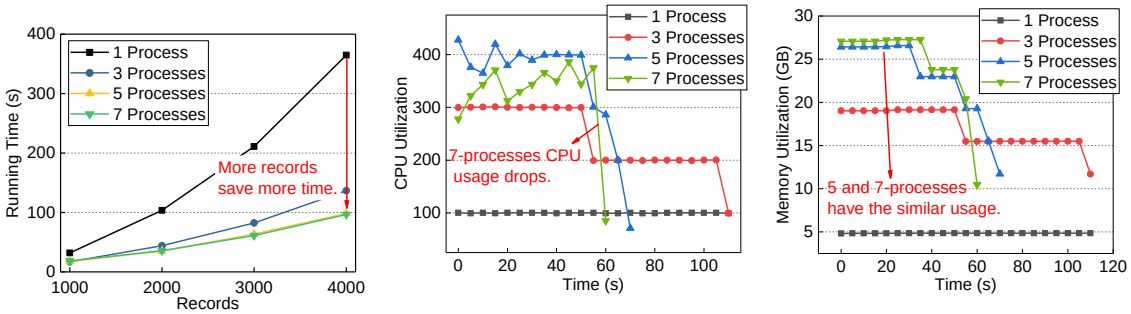
$$S_(i,j) = \frac{|N_(i) \cap N_(j)|}{\sqrt[2]{|N_(i)||N_(j)|}} \tag{3.3}$$

- Where S is the cosine-based similarity
- $N(i)$ and $N(j)$ mean the total number of users who rate 1 for item i and item j, respectively

By considering the counting of number of visits for the items, we come up with the adjusted Cosine Similarity as:

$$S_{(i,j)} = \frac{\sum_{u \in U}(R_{(u,i)} - \overline{R}_u)(R_{(u,j)} - \overline{R}_u)}{\sqrt[2]{\sum_{u \in U}((R_{(u,i)} - \overline{R}_u)^2)}\sqrt[2]{\sum_{u \in U}((R_{(u,j)} - \overline{R}_u)^2)}} \tag{3.4}$$

- Where S is the cosine-based similarity
- U stands for the group of users who rate both item i and item j
- $R_{(u,i)}$ or $R_{(u,j)}$ means the score for such the item by the user
- $\overline{R}_u$ is the average of the u-th user's ratings



(a) The Running Time Comparison.

(b) The CPU Usage.

(c) The Memory Usage.

Figure 3.3: The running time latency, the CPU usage for 4,000 sample records, and the memory usage for 4,000 sample records.

Global Relevance (GR)

The third method we proposed to use in the `dPanther` system is the Global Relevance (GR) recommender. The original GR is simply defined as:

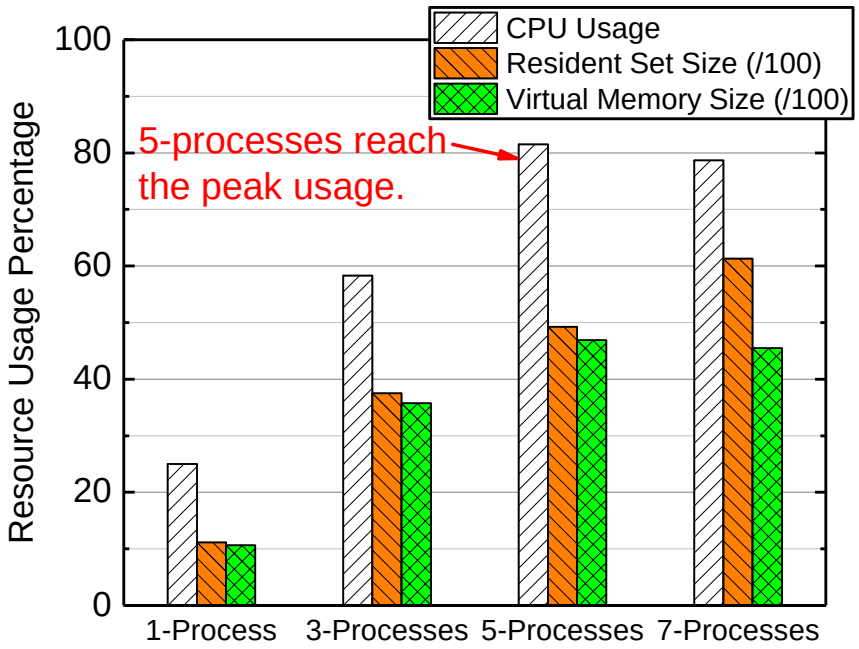


Figure 3.4: The Average Usage of CPU, Memory and Virtual Memory by using 1-process, 3-processes, 5-processes, and 7-processes for the task of running 4000 records.

$$GRScore \propto number\ of\ hits\ of\ the\ item \tag{3.5}$$

However, the collections in `dPanther` are generally consistent with geo-registration features, which appeal to users in specific regions rather than general interests. Therefore, we introduce enhanced GR scores with the consideration of the contribution as follows:

$$GR_u \propto N_i \quad where \quad L_i \in L_u \tag{3.6}$$

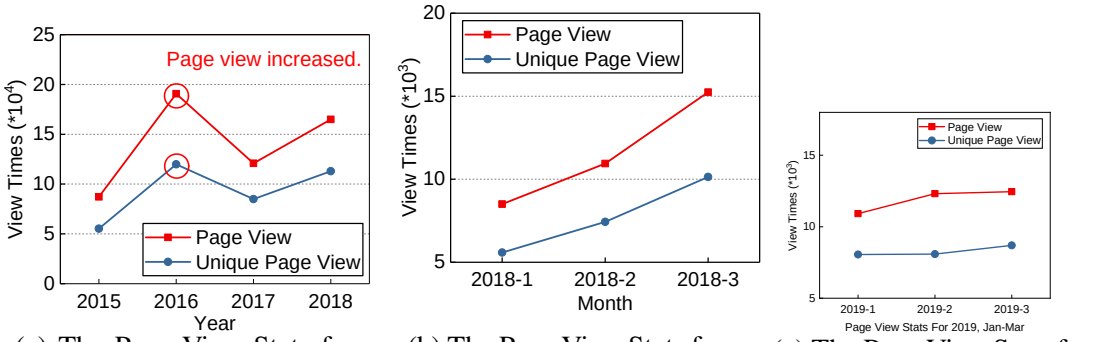
- Where *GR* is the Global Reference Score
- u is the current user
- i is the item in `dPanther` system
- N is the number of hits of item i
- L is the location information

Term Suggestion (TS)/Query Suggestion (QS)

The last method which we implemented is Term Suggestion (TS)/Query Suggestion (QS). Since one of the major components is metadata search/query function, many unsuccessful visits are caused by the search function failures. The suggestion function contains two steps. Firstly, the system suggests the key word based on the users' input. After a user hits the search, the system conducts a similarity analysis between the input search term and the subject key words from the metadata and then provides the suggestion. The similarity of the term to the metadata is generated by using the Jaccard index as suggestion by [LSM14]:

$$S_{(a,b)} = \frac{|DS_a \cap DS_b|}{|DS_a \cup DS_b|} \tag{3.7}$$

- Where S is the Similarity Score between the search term and the item subject key-words
- a is the input search term
- b is the item in the system
- DS is the data set that consists of the key works

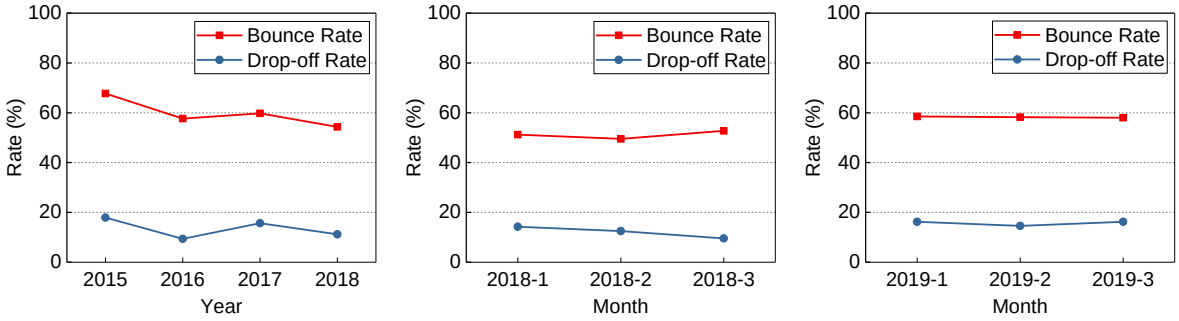


(a) The Page View Stats for the year 2015-2018.

(b) The Page View Stats from Jan. to Mar., 2018.

(c) The Page View Stats from Jan. to Mar., 2019

Figure 3.5: The Page View Stats for the year from 2015 to 2018, from January to March in year 2018, and from January to March in year 2019.



(a) The Bounce & Drop-off Rate from the Year 2015-2018.

(b) The Bounce & Drop-off Rate from Jan. to Mar., 2018.

(c) The Bounce & Drop-off Rate from Jan. to Mar., 2019

Figure 3.6: The Bounce & Drop-off Rate for the year from 2015 to 2018, from January to March in the year 2018, and from January to March in the year 2019.

3.5 Evaluations

We evaluate `dpSmart` from two different perspectives: computational complexity and the actual impact to its hosting Digital Repository Systems `dPanther`. The experimental evaluations answer the following questions:

- *Whether the multiple-process programming algorithm improves the hosting Digital Repository Systems performance?*
- *When it is integrated into the Digital Repository Systems, what are the benefits of `dpSmart` regarding the page views, bounce & drop-off rate?*
- *How are the Digital Repository Systems usability like Avg. Time on Page, Avg. Pageload, Avg. Direction Time, Avg. Server Response Time, and Avg. Page Overload Time improved by integrating `dpSmart`?*

3.5.1 Experiment Setup and Data Collection

Multi-process Programming Experiment

The experiments were conducted on a windows machine with four core CPU and 32 GB of memory. 4,000 metadata records are randomly selected from the database and processed to generate the item vector. We conducted the experiment by running the CB filtering algorithm against the 4,000 records by using 1-process, 3-processes, 5-processes, and 7-processes respectively. The algorithm run time, CPU Usage, and Memory Usage are collected.

System Data Collection

For the system evaluation, since `dPanther` is an existing production system, we focus on the real system statistics to evaluate if the recommendation module has a positive impact on the hosting system. Common performance metrics include Page View, Unique Page View, Average Time on Page, Bounce Rate, Drop-off Rate, Avg.Time on Page, Avg. Page Load Time, Avg. Direction Time, and Avg. Page Overload Time is collected for the evaluation. There are some milestones for the evaluation. The first module implemented in the system is the Customized Recommenders Module, which was published for production in January 2016. The Automatic Web Server Log Mining Module, User-group clustering Module, and Recommendation Strategy Module are related to each other. Therefore, these three modules were implemented all at once in January 2019. In addition, we also applied the optimized multiple processes programming so that we could rebuild the model more frequently. In order to evaluate the impact of these three modules, we compared the system statistics from January, February, and March of the year 2018 and year 2019. We rebuilt the model every week in the month of January and February and left the model as

is in March. The statistics for this period are also analyzed for the system changes after the new implementation.

3.5.2 Multi-process Programming Evaluation

As shown in Figure red3.3a, the total run time for the 4,000 record process has been reduced from 365 seconds to 137 seconds for 1 process to 3 processes and further reduced to 98 seconds after increasing to 5 processes. But the total run time is only reduced from 98 seconds to 97 seconds after we increased it to 7 processes. The figure also indicates that the more records processed, the more time is saved.

The CPU and Memory Usage over time in Figure red3.3b and Figure red3.3c also indicates that 5 processes approach best utilized the resources. Figure red3.4 also reveals that the average CPU Usage and Memory Usage reduced from 81.5% to 78.7% and 0.47% to 0.45% respectively.

3.5.3 Impact to the Hosting Digital Repository Systems

In Figure red3.5a, the page views and Unique Page Views have increased from 87,168 to 190,665 and 55,311 to 119,783 respectively from the year 2015 to the year 2016. This figure also reveals that both Page View and Unique Page View dropped in the year 2017 from 190,665 to 120,837 and 119,783 to 84,936, respectively. The reason may lie in that the model wasn't recalculated while new content and log data were generated. After recalculating the recommenders multiple times in the year 2018, we can see that the number increases again from 120,837 to 164,969 and 84,936 to 112,972 respectively. In Figure red3.7, the Avg. Time on Page also indicates that in 2015, users spent more time on average on a page, which is most likely from internal or academic users. When

the recommenders work and attract more general/layman users, the Avg. Time on Page dropped accordingly.

Figure red3.6a shows the Bounce Rate and Drop-off Rate trend from the year 2015 to the year 2018. Similar to the Page View statistics, both Bounce Rate and Drop-off Rate show a reverse proportional relation to the recommenders module. Even though this is not as clear as the Page View statistic, it still shows that an updated recommenders module can help to reduce the Drop-off rate. However, as shown in Figure red3.6b and Figure red3.6c, the same trend cannot be confirmed after implementing the Automatic Web Server Log Mining Modules, User-group clustering Module, and Recommendation Strategy Module. This is mainly because the stereotyping recommender is more used to enhance the recommendation quality and reduce the recommendation noise. It does not necessarily have a direct impact on the Bounce Rate and Drop-off Rate.

From Figure red3.7 and Figure red3.8, we can verify that the system usability does not decrease a lot compared to the year 2018. The Avg. Page Load Time, Avg. Direction Time, Avg. Server Response Time and Avg. Page Overload Time is increased compared to the year 2018, but still much lower than in the year 2016 and year 2017. Secondly, the Page View in Figure red3.5b and Figure red3.5c also increased in January and February in the year 2019 from 8,502 to 10,916 and from 10,942 to 12,314 respectively. The Page View has decreased in March from 15,239 to 12,450 compared to the year 2018 which caused by the out of dated model.

In summary, `dpSmart` has remedied the computational complexity problem by implementing the multi-process programming. The system statistics also confirmed that `dpSmart` effectively increased the Page Views of the hosting system by implementing the customized recommenders.

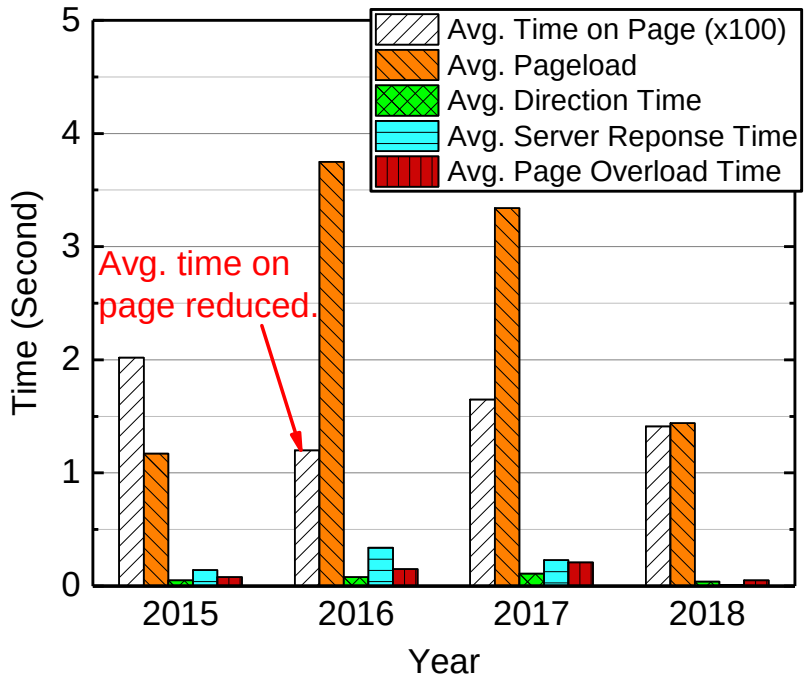


Figure 3.7: The system usability statistics from 2015 to 2018.

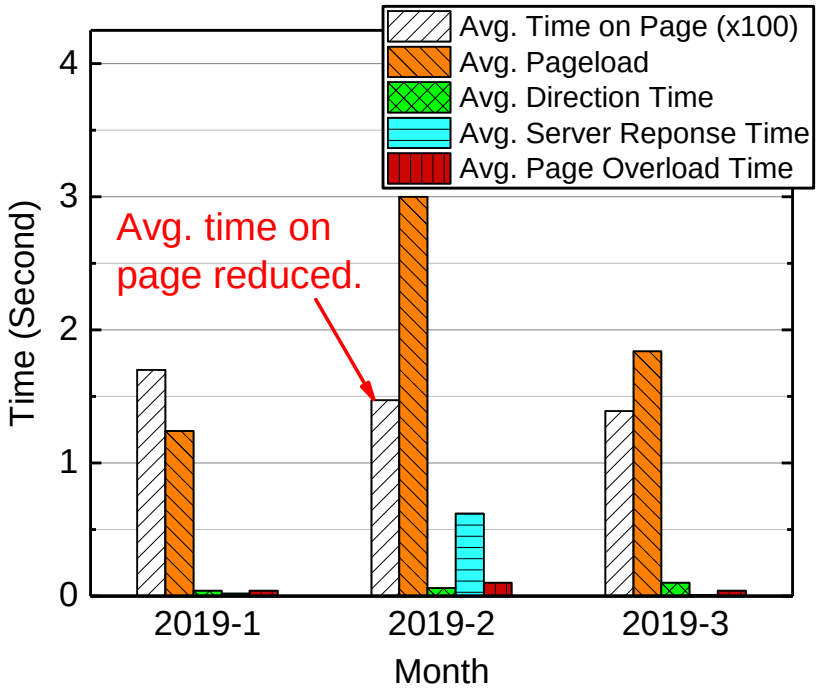


Figure 3.8: The system usability statistics from Jan. to Mar., 2019.

3.6 Conclusion

In this paper, we design and implement a flexible group-based recommendation framework, called dpSmart, and integrate it into a real-world Digital Repository System dPanther (http://dpanther.fiu.edu). The goal of dpSmart is to address the limitations of the digital repository system which include low discoverability, poor usability, and high drop-off visit rates. dpSmart provides the capability of automatic user vector generation from log data, clustering-based user group identification, flexible recommendation strategy, and multiple recommenders integration. The proposed system also suggests several customizing methods to customize the specific recommendation algorithms for the specific system. The experimental evaluation shows that by applying multi-process programming, the model-building time can be significantly reduced. The system usage statistics also indicate that during the evaluation time from January 2019 to February 2019, the Page Views have increased from 8,502 to 10,916 and 10,942 to 12,314 respectively, compared to 2018, demonstrating the effectiveness of our proposed framework.

CHAPTER 4

STRAIGHTLINE: FROM-DEVELOPMENT-TO-DEPLOYMENT MULTIPLE RESOURCES-AWARE MACHINE LEARNING APPLICATION PIPELINE

4.1 Introduction

ML has recently evolved from a small field of academic research to an applied field. According to McKinsey's global survey, ML is largely deployed in standard business processes and applications with nearly 25 percent year-over-year growth [CCH19]. For Industry 4.0, devices and machines across factories work on ML-based anomaly detection, edge robotics, or other industry-related applications [IBM21, HLL+19]. For automatic vehicles, numerous ML-based applications, such as object tracking, object detection, driving decision, and other autonomous driving-related applications emerge [KGS+21, PTAJ21, ZWH+21].

The life cycle of ML applications can be categorized into two stages: Model development and Model deployment. In model development, ML application developers need to go over three major phases: (i) data management: preparing data that is needed to build a ML model; (ii) model training: model selection and training (iii) model verification: to ensure the model adheres to certain functional and performance requirements. In model deployment, developers need to consider two major phases: (i) infrastructure building: building the infrastructure to run the ML model; (ii) model implementation: implementing the ML model itself in a form that can be consumed and supported. In different phases and stages, the requirements of resources are different. However, most of research focuses on one single stage or phase in the life cycle of ML applications. Few studies focus on from-development-to-deployment ML application pipeline. We face significant challenges in developing such a pipeline.

The first challenge is: *how to provision appropriate resources for different stages or phases?* In model training, GPU are indispensable to accelerate training process, whereas in model verification it is appropriate to allocate light-weight GPU resources to execute model inference. However, modern ML development systems, such as DB-Pal [WUG+20], Gpipe [HCB+19] and deployment system, such as MLflow [ZCD+18], MLCapsule [HZG+21], and other ML application-specific systems [KKS+19, RSR+21, CVZA20, MIW22, GXG+22], aim at optimizing one single stage or phase in the life cycle of ML applications. The second challenge is: *how to satisfy heterogeneous ML applications?* Since more and more ML models will be embedded in business processes and applications, different ML applications adhere to different computing environments and resources. However, existing studies assume that all ML models can be run and implemented in a unified environment [ZZL+22, ZWZ+21, CROS21, APYS20, ZYY+20, GKA+20], where the model inference is assumed to be done in one single and simple computing environment. In real-world industrial environments, it is intractable to have one permanent computing unit to serve all the ML applications. The third challenge is: *how to adapt to the hybrid infrastructure?* In real-world industrial deployment of ML applications, the infrastructure usually consists of different computing resources. However, existing works focus on one single computing resource, such as serverless computing [JCXL21, MD22], RESTFUL API [VVMH20, MLB+21, JLH+21]. In this case, some ML applications will face severe problems in satisfying the requirements of latency and response time if run with one single computing resource.

In this paper, we present StraightLine, a from-development-to-deployment multiple resources-aware ML pipeline, to address the challenges listed above. The key innovation is that StraightLine incorporates entire life cycle of ML applications and multiple computing resources into one pipeline. In sharp contrast to existing development and deployment systems, StraightLine offers from-development-to-deployment and multi-resource

implementation and placement, thereby greatly narrow the engineering gap between development and deployment.

We make the following contributions in the paper.

First, we study the existing development and deploy systems and discuss their limitations in the from-development-to-deployment and multi-resource design. To our best knowledge, we are the first to explore the possibility of the from-development-to-deployment and multi-resource design (Section 4.3).

Second, we design a novel model development abstraction to containerize the phases in model development. The main advantage of the containerization is that the provisioned resources for different phases can be done only when the execution of phases is started. There is no to provision the resources in advance. This design allows StraightLine to be adaptive to different requirements in different phases (Section 4.3).

Third, we offer different approaches to implementing ML applications, from RESTful APIs, serverless applications, and Docker containers. We also design an online placement algorithm to determine the placement of different ML applications in order to minimize latency and response time. This allows StraightLine to be adaptive to heterogeneous ML applications and hybrid infrastructure (Section 4.3).

Finally, we demonstrate StraightLine's training time in model development, session length, response time, and failure rate in model deployment (Section 4.4).

The rest of the paper is organized as follows: Section 4.2 summarizes the background and motivation of this paper. Section 4.3 elaborates the design of StraightLine. Section 4.4 reports the experiment results of StraightLine. Lastly, section 4.5 concludes this paper and discusses the future work.

4.2 Background and Literature Review

In this section, we first summarize the life cycle of ML applications in 4.2.1, and then review existing ML applications systems to understand the gap between development and deployment of ML applications in Section 4.2.2.

4.2.1 Background

Figure 4.1 shows the life cycle of ML applications. Model development and model deployment are two major stages. **Model development**. Model development consists of the following three phases: (i) data management, (ii) model learning and (iii) model verification [ACP21]. Data is an integral part of any ML applications. The overall predictive performance of ML applications largely depends on the training and test data as much as on the learning algorithms. The process of producing quality datasets is usually the very first stage in any production ML pipeline. Since data management and collection lack a standard approach and measurement, it is assumed that the data management can be implemented as a data pipeline by big data generators, such as sensors, log collectors, and transaction data collectors. Model learning is the phase of model development that draws the most attention within the ML community. Generally speaking, model is selected to be used to prove the concept of the proposed ML solution and get the end-to-end setup in place [WDD+19]. Then, the chosen model is fed with a collected dataset in order to learn certain patterns or representations of the data in the hybrid infrastructure of in-house data center, local servers, or docker containers. Lastly, model verification is regarded as a fundamental step to ensures the quality of the product and reduces maintenance costs. ML models are expected to generalize well to unseen data, demonstrate reasonable performance on edge cases and overall robustness, as well as reach specified functional requirements [PUL20].

Model deployment. Model deployment consists of two major phases: (i) infrastructure building and (ii) model implementation. In real-world industrial environments, the software infrastructure is hybrid, which includes in-house data centers, local servers, docker containers, serverless platforms (e.g., AWS Lambda [Amab]), to provide lower inference latency and response time. Such a hybrid infrastructure offers a variety of ML services and implementations in a collaborative fashion, whereas model implementation mainly focuses on how the ML applications are implemented in and incorporated into the infrastructure to shorten latency or response time. Traditionally, a trained ML model can serve as a RESTful API [SAM15] or a lightweight web server, Flask [fla]. Moreover, enterprise-level solutions and open-source solutions, such as Microsoft's ML.NET [mln] and Deep Java Library [DJL] are also available. For small or middle-size companies, the on-demand cloud computing ML platforms, such as AWS ML [AWS] and Azure ML REST API [mic] are also the options for ML implementations. Recently, Docker containers [And15] and serverless computing [BCC+17] offer more portable options since ML applications usually require different operating systems, ML application runtime (e.g., TensorFlow [ABC+16], PyTorch [PGM+19], PyWren [JPV+17], etc.), and language environment like Python, Java, or R. Docker containers and serverless computing offer plug-and-go computing services since they can connect target environments automatically.

4.2.2 Existing ML Application Pipelines

Elshawi et al. presented a framework that includes the basic clouding computing resource (e.g., storage, CPU, and memory), data streaming platforms (e.g., Hadoop [had], Spark [ZCD+12], Flink [CKE+15]) and workflow environment (e.g., Spark [ZCD+12], Google ML [goo], TensorFlow [ABC+16], KeyStone ML [key]). The proposed framework sup-

ports the ML application workflow from data management to model training while it does not support ML application services [ESTT18].

Wang et al. presented Rafiki, an ML analytics service system to provide the training and inference service of ML models. Howerver, Rafiki is limited to 3-node cluster so it is not applicable to hybrid infrastructure [WWG+18].

Sánchez-Artigas et al. proposed a Serverless enabled ML system, MLLess, which claims 15 times faster than serverful ML application and successfully reduces the cost. The paper compared the ML model training process in detail by using PyTorth and PyWren on local data center and serverless environment respectively [SAS21]. However, the experiment is based on the assumption that all the training environments use CPU clusters since the GPU cluster is not supported by Lambda. This generates a critical bias from the real world applications since GPU cluster has become a fundamental requirement for ML applications.

Naranjo et al. proposed a serverless gateway for event-driven ML inference in multiple clouds. This paper presented a framework that allows to package the inference function as a RESTful API on both serverless and on-premise cloud [NRMB21]. However, the paper focuses more on the feasibility of the proposed the framework. The performance difference between the on-premise and serverless resources is not discussed. The paper also focuses on the inference function based on pre-trained model. The model development is overlooked even an AWS Batch is included in the framework which can provide GPU cluster support for the model training.

Shukla et al. proposed an anomaly detection application in for Internet-of-Things (IoT) devices over edge computing networks. The data is collected by IoT sensors and sent to the edge for model training and deployment. For model inference, IoT devices send data to edge for model inference and the edge sends the inference results to the cloud

for analytics [SS22]. However, the paper only focuses on single edge-based environment while ignoring the hybrid infrastructure.

Most of works focus on one single stages or phases in the life cycle of ML applications. Moreover, most of works assume that the infrastructure for ML applications is unified and monotonous. This serves as the motivation of this paper to offer a from-development-to-deployment system for ML applications.

4.3 Design

4.3.1 Overview

StraightLine aims to achieve the following goals:

- From-development-to-deployment design. It defines clear modules in the life cycle of ML applications.
- Adaptability. It designs multiple implementations adaptive to heterogeneous ML applications.
- Low latency. It achieves low latency for ML applications in the hybrid infrastructure.

As shown in Figure 4.2, StraightLine consists of three layers: *model development abstraction*, *multiple implementation deployment*, *real-time resource placement*.

Layer 1: Model development abstraction. All phases in model development are containerized into multiple different containers based on available GPU resources. For the phases of data management and model training, we build up powerful NVIDIA-Docker [NVI] to offer plug-and-go provisioning instead of using a traditional GPU cluster. For model verification, we build up a lightweight NVIDIA-Docker to verify the model per-

formance. The final trained and verified ML models will be compressed and serve as the implementational fundamentals for ML applications.

Layer 2: Multiple implementation deployment. Based on the compressed ML models, we build up corresponding RESTful APIs, serverless applications, and Docker containers. The key innovation is to leverage Docker containers to adapt to different computing environments.

Layer 3: Real-time resource placement. Based on the different-scale requests from ML applications and time-varying resources in the hybrid infrastructure, we design an online resource placement algorithm to determine the optimal resource placement for requested ML applications in order to minimize latency and response time.

4.3.2 Model Development Abstraction

The stage of model development usually demonstrates the requirements of high speed I/O, powerful CPU processor, high memory, GPU-preferred yet not always-on environment. Moreover, this stage usually demands heavy computational resources for a very long period of time so the stability is an important factor. For example, it takes the CPU of Intel i9-7890 AT 10 hours to finish the training task with Xception [Cho17] on a small-scale animal recognition dataset. Therefore, the challenge is *how to provision appropriate resources for different phases in model development?* We need to allocate different resources for different phases in model development, and when the allocated resources should be released back to the infrastructure automatically after tasks finish.

To achieve plug-and-go and stable functionalities in model development, we containerize the phases with NVIDIA-Docker [NVI]. NVIDIA-Docker is a thin wrapper on top of docker [NVI]. When creating a container using NVIDIA-Docker, we specify the information of the CUDA devices, volumes, and libraries in NVIDIA-Docker and it creates

a container with this information. This allows us to have a plug-in GPU-aware container to implement high-performance and stable model development. Moreover, when the running tasks finish, the provisioned resources will be released back to the infrastructure automatically.

In practice, we specify more GPU resources in NVIDIA-Docker for data management and model training, whereas specifying little GPU resources for model verification since model inference is light weight compared to model training in terms of computation complexity. After model development is completed, the trained model will be compressed as a `.h5` file and go through the stage of model deployment.

4.3.3 Multiple Implementation Deployment

StraightLine is designed for a hybrid infrastructure so the compressed model will be implemented in three different ways: 1) local web server, 2) RESTful APIs, or 3) serverless computing. However, it is possible that the hybrid infrastructure cannot offer a compatible environment to many heterogeneous ML applications. Therefore, the challenge is *how to further improve the flexibility of the infrastructure?* Each computing unit in the hybrid infrastructure may run different operating systems, ML application runtime (e.g., TensorFlow [ABC+16], PyTorch [PGM+19], PyWren [JPV+17], etc.), and language environments (e.g., Python, Java, or R). It is necessary to consider the implementation difficulty resulted from software environment conflicts.

We further offer the implementation of containerized ML applications. As shown in Figure 4.3, a containerized ML application only contains core information (e.g., model weights, and inference requirements) and the target environment (e.g., ML application runtime and language environment). Once a containerized ML application is trigger in the infrastructure, it can connect to the specified target environment and resources. When

the task is finished, the provisioned resources will be released back to the infrastructure. Moreover, we can execute cross-platform ML implementation by specifying different target environments, such as different versions of Linux, Windows or serverless environments.

In practice, we use the Flask [fla] Python library to implement RESTful APIs of ML implementation since most of machine learning library is built on Python. For serverless computing, we use AMS Lambda [Amab] to implement ML applications.

4.3.4 Real-time Resource Placement

In the real-world industrial infrastructure, multiple computing resources are available. Moreover, uncertainties, such as request frequency, and request data size, arise in running ML applications. A vivid example is that a single image classification request can be done by Flask API while a batch of 100 image classification simultaneous requests needs to be sent to Amazon Lambda to handle. Therefore, the challenge is *how to allocate appreciate resources according to the requested ML applications?*

To allocate resources for upcoming ML requests, we use web servers to collect two different features: (i) request frequency, and (ii) request data size. Specifically, request frequency represent how many requests are received in a particular period of time by the server end. Request data sizes show the data sizes of the entire request. In practice, the request frequency will be recorded in the web server log and the request data sizes can be detected by the web server.

Suppose that there are a set of requests R in the waiting queue. Denote request id and request data size of request $r \in R$, request frequency at time t by r_{id}, r_d, f_t, respectively. Also, let F and D denote the request frequency and request data size thresholds, respec-

tively. To find the best possible implementation for the upcoming ML application, we propose dynamic placing algorithm in Algorithm 1.

In line 5, we first investigate frequency and data size of the request. If current request frequency is larger than the frequency threshold and data size is smaller than the data size threshold, we implement the request with serverless computing. It is because the web server should have born heavy load and small data sizes bring small communication overhead to transmit request to serverless resources. Therefore, it is appropriate to offload the request to serverless resources. In line 8, we consider deploy the request with large data size in Docker since the requests with large data sizes usually tolerate longer response time. For example, the analysis of high-resolution medical images can tolerate higher response time. It is appropriate to put the requests in the queue of Docker. In line 11, we implement the requests with Flask if the request has moderate data size and the current request frequency is low. It is because such a request matches the characteristics of Flask: faster but unable to process requests with too high request frequencies and data sizes. In lines 14 and 17, only when the Flask is not available for more requests and the request frequency is moderate, are the requests with moderate data sizes processed by Docker and serverless, where docker is prioritized. The requests allocated with Docker container will be run with RESTful APIs.

4.4 Evaluation

We evaluate StraightLine on a real hybrid infrastructure testbed. We explore its performance for real-world ML application request. Our evaluation answers these questions:

- Does StraightLine reduce training time and achieve equivalent model accuracy to CPU cluster?
- How does StraightLine perform in RESTful APIs, serverless computing, and docker?

Algorithm 1: Dynamic Placing Algorithm

Input: A set of requests R, and each request r's request id r_{id} request frequency at time t f_t, request data size r_d, frequency threshold F, data size threshold D, available resources of Flask S_F, and Docker S_D.
Output: The placement K of the set of requests R.
$K \leftarrow \emptyset$;
for $r \in R$ **do**
    if $f_t > F$ *and* $r_d < D$ **then**
        $K \leftarrow K \cup \{(r_{id}, \texttt{serverless})\}$;
        /* Run request r in serverless computing */
    else
        if $r_d > D$ **then**
            $K \leftarrow K \cup \{(r_{id}, \texttt{docker})\}$;
            /* Run request r in docker */
        else
            if S_F *is not empty* **then**
                $K \leftarrow K \cup \{(r_{id}, \texttt{Flask})\}$;
                /* Run request r in Flask */
            else
                if S_D *is not empty* **then**
                    $K \leftarrow K \cup \{(r_{id}, \texttt{docker})\}$;
                    /* Run request r in docker */
                else
                    $K \leftarrow K \cup \{(r_{id}, \texttt{serverless})\}$;
                    /* Run request r in serverless computing */
                end
            end
        end
    end
end

- Does online placement algorithm in StraightLine improve latency and response time?

4.4.1 Setup

Real hardware. Table 4.1 specifies the real hardware for the experiments. For model development, we simulate an in-house data center using a 2-node GPU-ready in-house data center. Each node contains 36 cores of Intel(R) Core(TM) i9-7980XE CPU at 2.6Hz, 64GB of memory, and a NVIDIA GeFore GTX 1080 Ti GPU card with 11GB of dedicated GPU memory. NVIDIA-Docker's equipment is similar to the in-house data center but only has 36 cores CPU, 64GB memory, and 1 GPU resource. For model deployment, we build up a hybrid infrastructure that consists of a local web server with 12 cores of Intel(R) Xeon(R) E-2176M CPU at 2.70GHz, 32GB of memory and serverless computing of AWS Lambda [Amab] with `us-east2` and 5GM memory.

Emulation deployment. For RESTful service, we use TensorFlow [ABC+16] as the ML application runtime and Flask [fla] to expose it as the RESTful APIs. The web server is the Internet Information Server (IIS)[1] hosted on a Windows Server 2012. The WFastCgi module[2] is used to bridge the Flask API and the IIS server. The Flask application is setup as a virtual site and the actual inference function is set as a route of the virtual site. We set the CPU and memory share in the application pool as 100% so it can use up all the resources. For the network, we set the connectionTiemout as 5 minutes, maxBandwidth and maxConnections as 4GB, and maxURLSegments with 32 pieces.

For serverless computing, we use AWS Lambda [Amab] and AWS API Gateway [Amaa] for the ML application serverless implementation. The applications are set up under `x86_64` architecture with python 3.9.1 environment. The inference function is coded

[1] https://www.iis.net/

[2] https://pypi.org/project/wfastcgi/

as the Lambda function and a route `img_classify` is defined as a trigger in the API gateway. As AWS Lambda allows 128MB to 3GB for the provisioned memory, we set up two experimental environments with 2GB and 3GB memory. The timeout is set as 50 seconds which is the same as the local web server. The ML application is transferred as a docker container and the runtime of the TensorFlow Keras is defined inside the DockerFile.

For docker containers, we follow the the Dockerfiles on NVIDIA official repository[3] to launch different Linux systems. We build the docker environment inside the existing Window server stack to mimic the real-world scenario. First, we use Windows Subsystem for Linux (WSL2)[4] to set up a virtual environment in the existing Windows server. After the WSL2 successfully installed, we can setup the docker environment inside the Ubuntu instance. Second, we installed the NVIDIA Docker Toolkit from the official repository[5] so we can run ML task with docker. Third, we further modify the Docker files to make the tasks run automatically. We generated a `requirement.txt` file from the Python script we used for model training. We modified the existing Dockerfile by adding the RUN `pip install --no-cache-dir -r requirements.txt` and COPY in the Python script to the target directory. Lastly, we defined the starting command to run the Python script so that the docker container can automatically run the task when the image activate.

ML model and dataset. We choose an image classification mode, Xception [Cho17], as the base model to test in the different implementation environments. As shown in Table4.2, Xceptioin can reach up to 95% accuracy rate for image classification tasks and requires a minimum of 110.9MB for storage and the inference overhead is 109.4 ms. Xception model is suitable for our experiment since it demands intensive computational power in model development and its model size is also significantly larger than regular web applications. Therefore, we use it as the test base for our evaluation process. The

[3]https://github.com/NVIDIA/nvidia-docker

[4]https://learn.microsoft.com/en-us/windows/wsl/about

[5]https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html

experiment use an open-source training data set [xce] as shown in Figure 4.4, containing 299*299 4000 images and labeled as 1000 images for cats, 1000 images for chook, 1000 images for dogs and 1000 images for horses. The data set is split 80% (3200 out of 4000 images) into samples used to train the models and the rest (800 images) for validation.

Metrics. For model development, we focus on *running time, accuracy, loss, and confusion matrix* of the trained model. For model deployment, we care about *session length, response time, and failed rate* when implementing ML applications in different computing platforms.

Table 4.1: Summary of the computational resources for the experiment setup

Resouce	**Description**
In-house data center	72 CPU core, Inter(R) i9-7980XE AT 2.6Hz, 128GB memory, NVIDIA GeForce GTX 1080 Ti with 11GB GPU Memory *2
NVIDIA Docker	36 CPU core, Inter(R) i9-7980XE AT 2.6Hz, 64GB memory, NVIDIA GeForce GTX 1080 Ti with 11GB GPU Memory
Web Server	12 CPU core, ntel(R) Xeon(R) E-2176M CPU at 2.70GHz, 32GB memory
AWS Lambda	us-east2, 5GB Memory

Table 4.2: The official parameters of Xception [Cho17]

Model	**Size (MB)**	**Accuracy**	**Time (ms) per inference step (CPU)**	**Time (ms) per inference step (GPU)**
Xception [Cho17]	110.9	95%	109.4	8.1

4.4.2 Model Development

We measure the performance of StraightLine in model development. We adapt Google TensorFlow [ABC+16] to perform the training of the model. The entire GPU cluster is

set up as NVIDIA Docker [NVI] in order to realize the designed flexibility. A 20-fold cross-validation technique is used to evaluate the models during training. The training dataset is divided into 20 subsets: 16 are used for training and the remaining 4 are used for testing. This process is repeated until all samples of the training dataset have been used. The accuracy of the model is the average of the accuracy observed in each iteration. The same training task is performed on the same cluster with CPU and GPU respectively.

Running time. Figure 4.5 reports the running time of GPU cluster and CPU cluster. It is obvious to see that the CPU cluster takes 644.2 minutes, whereas the GPU only takes 32 minutes to complete the same task. Additionally, since the GPU cluster is a NVIDIA-Docker, the resources will be provisioned only when the task is triggered. This significantly reduce the cost while enhancing the performance. It is because StraightLine containerizes the phases in model development so StraightLine can connect required GPU resources and run the training tasks smoothly.

Confusion matrix. We further generate the confusion matrix to compare the the actual model performance for image classification tasks. Since we already know from the previous evaluation that both models reach the same accuracy rate and the end, we do not expect obvious difference. The result in the Figure 4.7 also supports this assumption.

Accuracy and loss. Figure 4.6 compares the CPU and GPU clusters' accuracy and loss. Apparently, after 20 epochs, both models reach 99% training accuracy and 70% - 80% validation accuracy and around 0.9 validation Loss. Algorithm CPU and GPU clusters can finis the training tasks, however, from chart **(a)** and **(b)**, we know that for the CPU cluster, it took 9-10 epochs to get the final stable accuracy and Loss. On the other hand, as shown in **(c)** and **(d)**, it only takes 1-2 epoch to reach a final and stable accuracy. Even though it took around 10 epochs to get the final Loss, the deviation for the Loss is obviously smaller than the CPU cluster.

4.4.3 Model Deployment

We set up an emulated hybrid infrastructure for the seamless implementation of ML applications among local servers, data centers, and AWS Lambda serverless environment. To simulate the request load, we use cloud-based performance scripting client, Artillery [art]. We implement ML application of Xception on different platforms and measure the performance with different request frequencies and request data sizes shown in Figure 4.8 are launched. We increase the load from 10 to 7000 within 180 seconds.

RESTful service. Figure 4.9 reports the load test results of Flask API implemented on the local web server and the in-house data center. From Figure 4.9a we see that the local web server and in-house data center achieve the similar failed rate when total sessions reaches 1300. From Figures 4.9b and 4.9c we can see that the session length in the local web server surges when total sessions reach 1200, whereas the session length in the in-house data center increases when total sessions reach 1400. It is because the web server implements single thread to run the ML application.

Serverless computing. Figure 4.10 reports the results of session length, response time of AWS Lambda stack implemented with 2GB and 3GB docker. It is obvious to see that the performance is improved significantly on AWS Lambda stack. We can see that the median response time stays around 300-500ms even when the request frequency increases up to 6000 per 180 seconds and the failed rate reaches up to 60% when the request frequency rises to 6000 per 180 seconds. The results on AWS Lambda outperforms the traditional RESTful API implementations.

Hybrid Infrastructure. Figure 4.11 shows the comparison of failed rate of running ML applications on different computing platforms. It is obvious indicate as the requests frequency increases, the serverless computing manage the failed rate in a acceptable level. In addition, as the provisioned memory increased from 2GB to 3GB for the severless computing, the failed rate decreased. Therefore, serverless computing tends to be a good

solution for high request frequencies environment. It is because serverless computing can offer unlimited resources for running ML applications as well as provide a efficient orchestration method to trade-off the required computational resources to the provisioned memory.

Figure 4.12 shows the comparison of response time of running ML applications on different computing platforms. Clearly, the Flask API outperforms the rest. It is because the Flask API can offer local computation for ML applications for application to directly call the inference function. Both Docker and AWS serverless solution cause longer response time because of the docker container activation overhead.

4.5 Conclusion and Future Work

Existing model development and model deployment systems were designed for one stage or phase in the life cycle of ML applications. In this paper, we present StraightLine, a from-development-to-deployment ML application pipeline that enables cross-platform ML applications and hybrid infrastructure implementations. StraightLine leverages docker to containerize the stages in model development to offer plug-and-go provisioning, and build up multiple computing platforms and cross-platform containers for model deployment. To consider the hybrid infrastructure, StraightLine designs an online resource placement algorithm to allocate computing resources for ML applications, using request frequencies and request data sizes as the input of the algorithm. An interesting question for future work is how to incorporate edge computing resources as part of the infrastructure. Edge computing is an emerging paradigm for mobile users since it offers proximity-aware services since the edge is usually located in the proximity of mobile users. Edge computing units are usually connected via wireless network so it is necessary to con-

sider the communication failure, possible packet loss. We will open source StraightLine, together with the data used to produce the results in this paper.

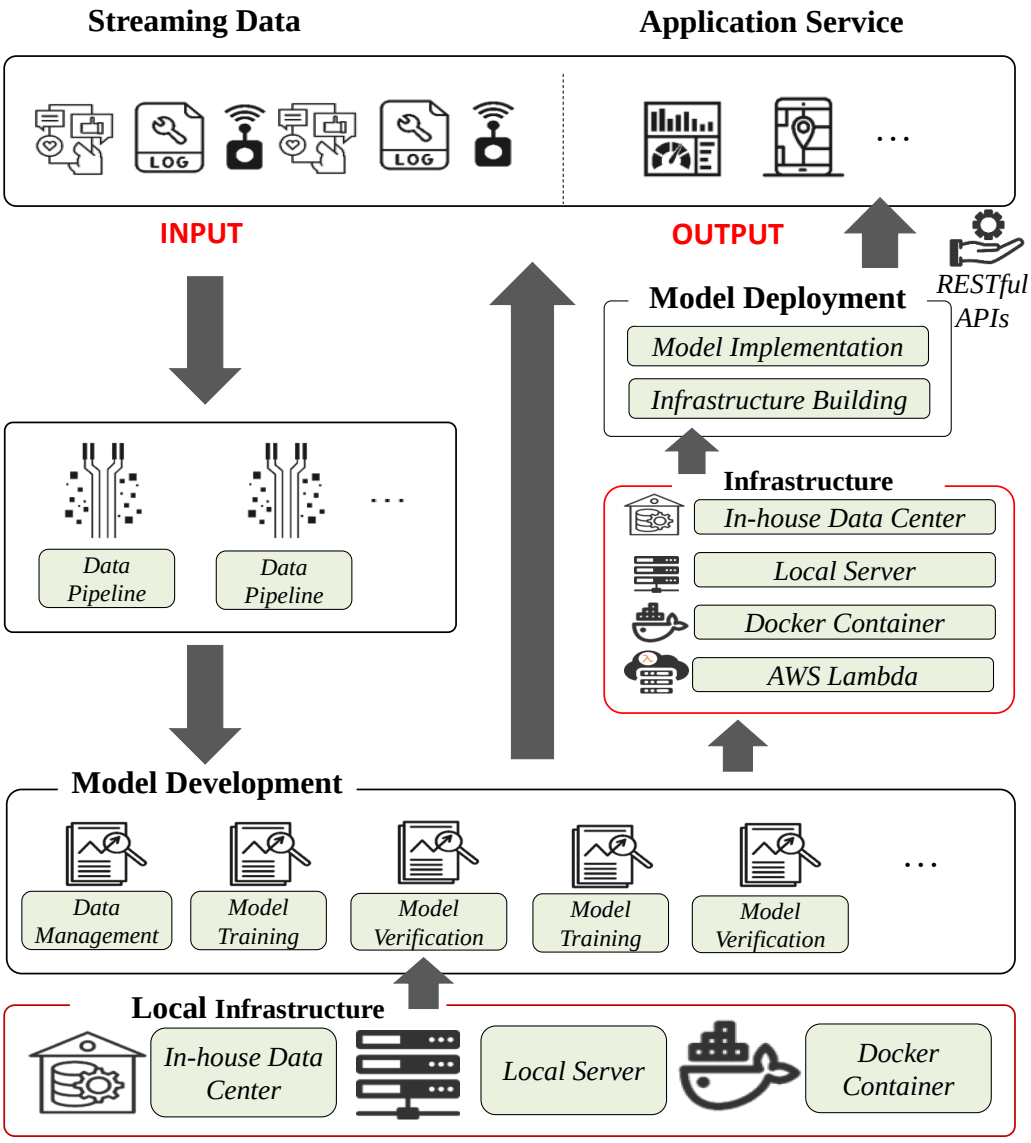


Figure 4.1: The conceptual life cycle of ML applications. In model development, the streaming data serves as the training data and goes over predefined data pipelines, and ML models are trained and verified with processed data using local infrastructure (e.g., in-house data center, local server, or docker containers). In model development, the trained ML models are then deployed on the infrastructure composed of different computing resources. Then, applications services send requests to the infrastructure through, for example, RESTful APIs.

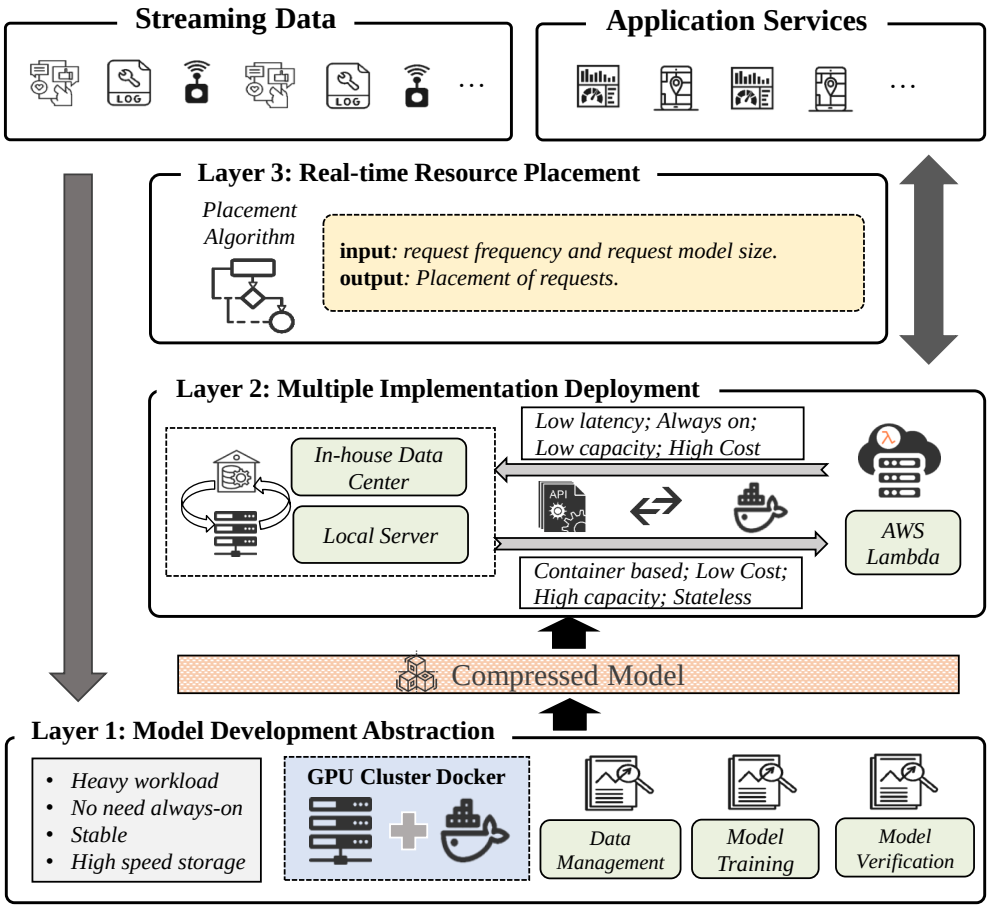


Figure 4.2: The workflow and three layers in StraightLine. In Layer 1, StraightLine uses NVIDIA-Docker to define the execution in model development. In Layer 2, StraightLine deploys multiple implementations for ML applications. In Layer 3, StraightLine runs the online resource placement algorithm to place computing resources for upcoming ML requests.

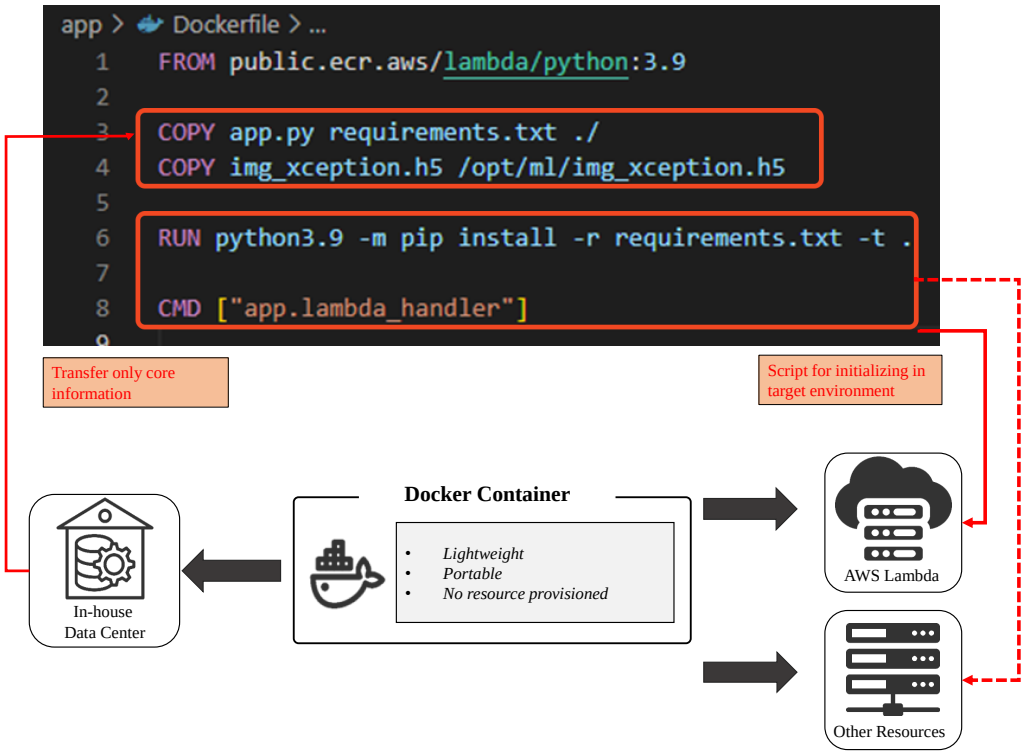
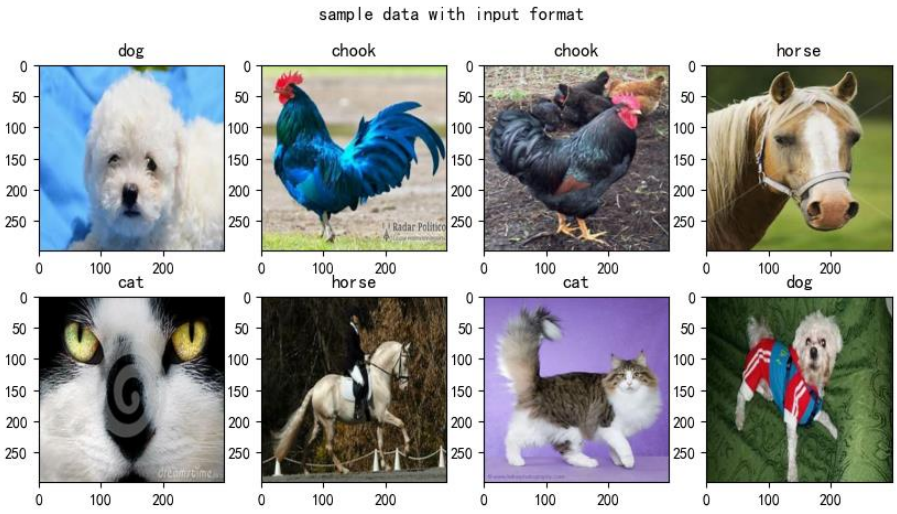


Figure 4.3: The core information in the dockerfile. It only copies over the necessary resources and the script to start up in the target environment. There is no operating system-related information carried out and no preset provisioning resources are required.




```
batch_size = 4
img_height = 299
img_width  = 299

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=12,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

✓ 0.4s

Figure 4.4: The sample training data set after pre-processing.

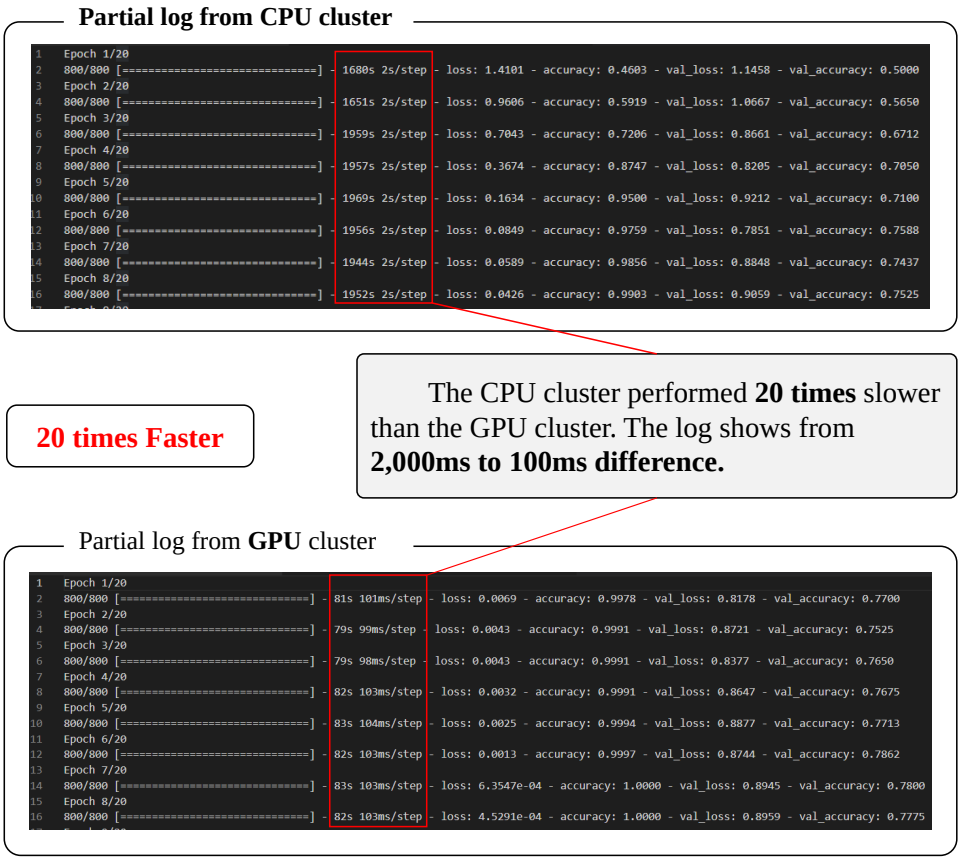
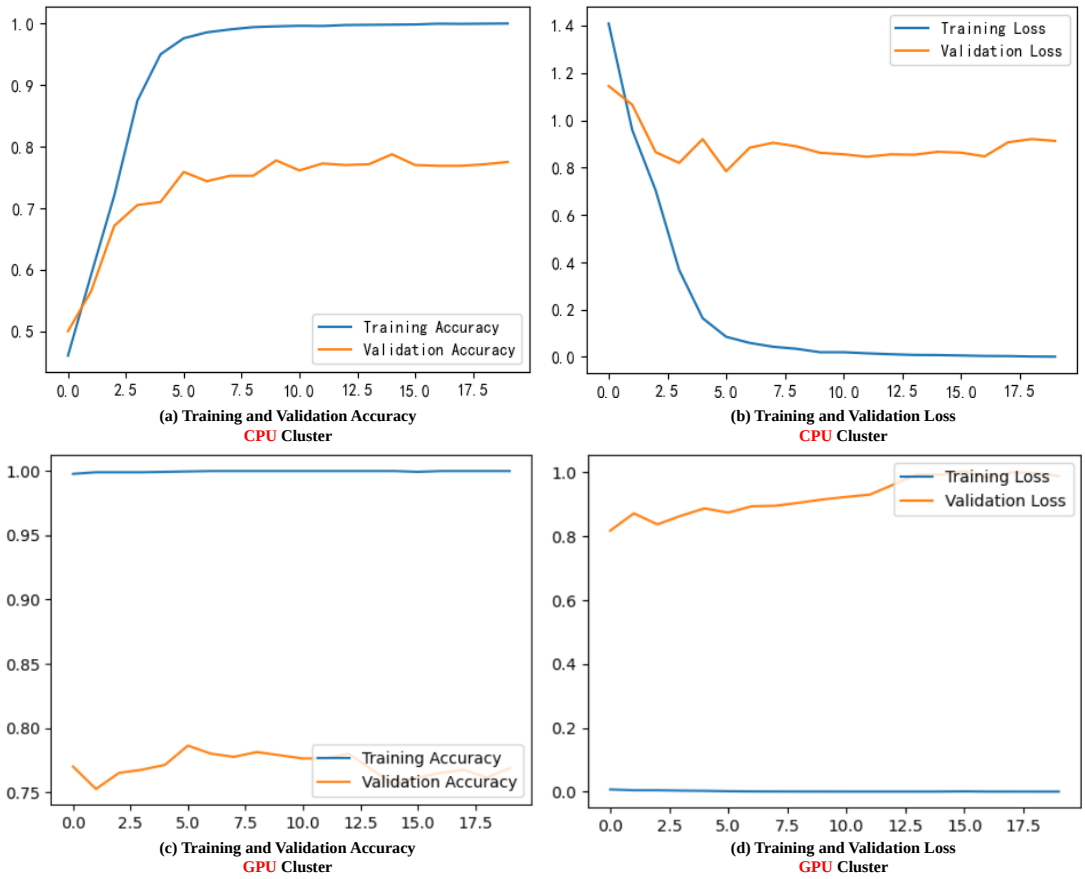


Figure 4.5: The training log demonstrate the significant improvement from the GPU cluster to the model training task. Each step the GPU cluster perform on average 20 times faster than CPU cluster



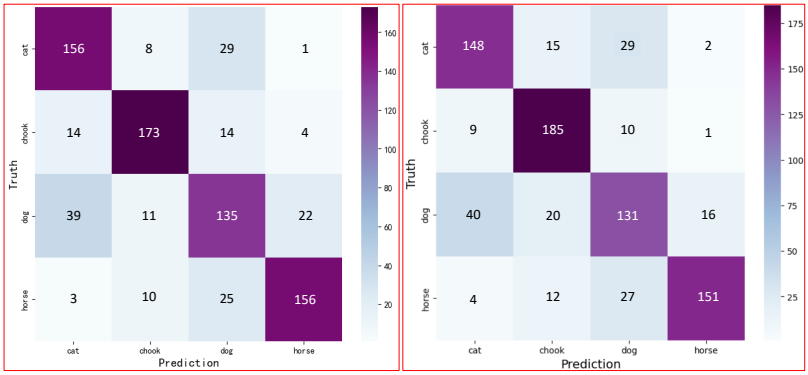
(a) Training and Validation Accuracy CPU Cluster

(b) Training and Validation Loss CPU Cluster

(c) Training and Validation Accuracy GPU Cluster

(d) Training and Validation Loss GPU Cluster

Figure 4.6: Training and validation Accuracy and Loss for the same task running by the CPU and GPU cluster respectively.



(a) Confusion Matrix for CPU Cluster

(b) Confusion Matrix for GPU Cluster

Figure 4.7: The confusion matrix generated by the model trained with the CPU cluster and the GPU cluster respectively. The result does not indicate obvious difference between two matrix.

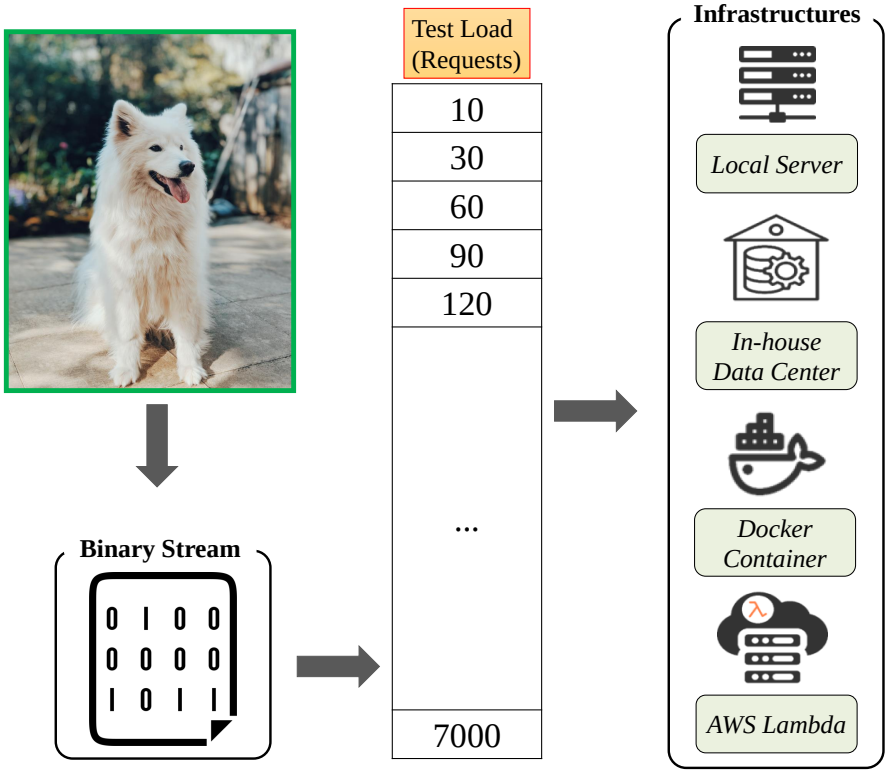
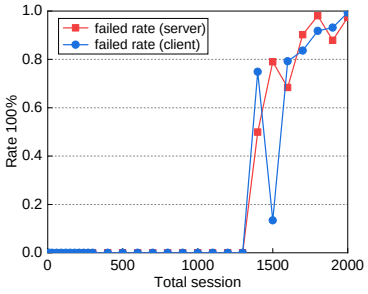
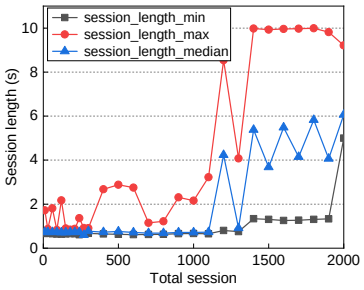


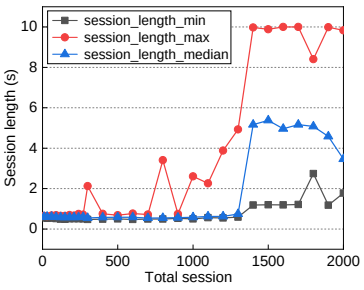
Figure 4.8: The experiment is designed for applying same batch of load test within 180 seconds with the same input image to different ML application implementations



(a) The failure rate on the local web server and in-house data center

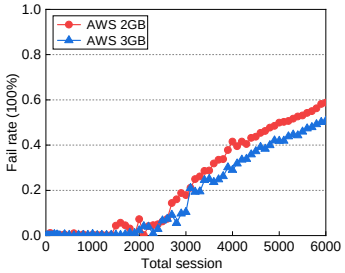


(b) The session length on local web server

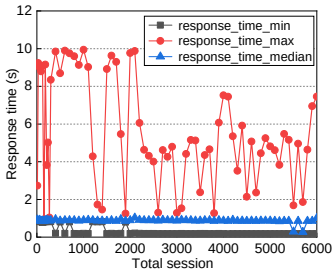


(c) The session length on in-house data center

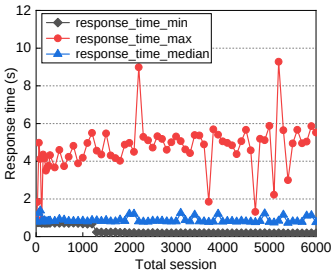
Figure 4.9: The performance of a Flask API server implemented on a local web server and in-house data center. **(a)** shows the failure rate of the local web server and the in-house data center. **(b)** and **(c)** show the session length of the local web server and the in-house data center, respectively. Total sessions start from 10 requests per 180 seconds to 2,000 requests per 180 seconds against the Xception model.



(a) The failed rate change on AWS Lambda 2GB and 3GB respectively



(b) Session Length and Response Time for AWS 2GB Memory Provisioned



(c) Session Length and Response Time for AWS 3GB Memory Provisioned

Figure 4.10: The graph shows the comparison for the same Xception application docker implemented on AWS Lambda with 2GB and 3GB provisioned respectively. **(a)** shows the failed rate directly impacted by the provisioned memory. **(b)** and **(c)** indicates the the application latency does not affect much by the load changes.

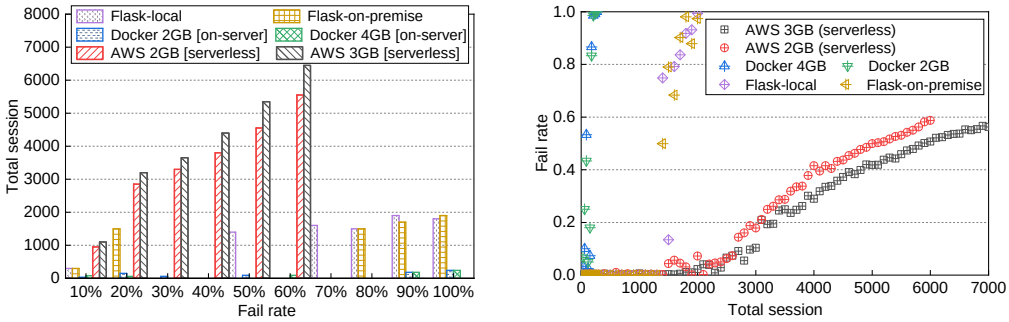


Figure 4.11: Results shows serverless implementation is dominating for the high frequency request environment. It significantly enhance the application capacity from 7-8 per second to 30 per second.

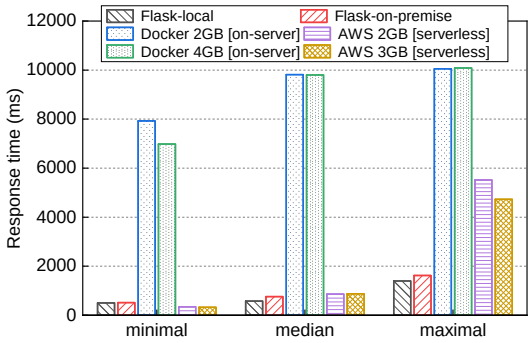


Figure 4.12: This figure shows the Flask API still out perform the rest implementation in terms of response time.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this dissertation, we focus on the long-tail problem of ML application implementation in various industries. In chapter 1, we present the motivation for this research is to solve the **domain-specific problem for Ml application development** and the **ML application implementation in heterogeneous infrastructure environment**. The contributions of this dissertation are:

- proposed dpSmart as a conceptual framework for ML application development in the domain-specific area.
- proposed StraightLine as a serverless enabled hybrid framework for Ml application implementation in the heterogeneous infrastructure environment.

In chapter 2, we conduct a background review of both ML application development for domain-specific areas and the Ml application implementation environments. For the domain-specific problem, recent research and similar studies are reviewed. The review result indicates a framework that can minimize the domain knowledge dependency is in demand. For the ML application implementation problem, we have reviewed the existing infrastructure from on-premise private, public cloud, hybrid cloud, and serverless computing. The review result revealed the need for a framework that can provide flexibility between different infrastructures.

In chapter 3, we demonstrate how to apply dpSmart from a conceptual framework to a physical implementation for a recommendation system in the digital library domain. The detailed design and an evaluation of the effectiveness and the system performance impact are presented in this chapter.

In chapter 4, we present StraightLine as a serverless computing enabled hybrid framework for ML application implementation. StraightLine provides the model development abstractions layer, multiple infrastructure deployment layers, and real-time resource placement layers. By introducing this layered model, different types of ML applications can be developed in a dedicated environment and deployed in a separate, hybrid, and serverless computing enabled environment. StraightLine also provides a real-time placement algorithm to optimize the ML application performance.

5.2 Lessons Learned

`lack of training data`. Getting the right training data means collecting or identifying the data that correlate with the outcomes we want to predict. However, obtaining rich training data is challenging because it takes a lot of time and domain-specific knowledge to find the data that contains a signal about events we are interested in

`Data sharing barriers.` Private data owners may be reluctant to share the data. Even when private data (e.g., administrative data, field treatment data) are shared, it is not certain that a private data set is compatible with another private data set or with other public data sets. For example, samples from public agencies are usually conducted according to certain statistical standards, whereas this is not necessarily the case when private data are collected.

`Theory and Practice Gaps` Empirical models trained using machine learning methods are difficult to generalize and used in domain-specific problems. For example, in many recommendation system research, the collaborative filter is considered a common recommender. However, when implemented in the digital library domain, it could produce more recommendation noises for academic users.

5.3 Broader Impact

The experimental approach used in this dissertation is by abstracting the problems from real-world projects, making a hypothesis, and verifying from a simulated environment. This approach can make a broader impact since it can be generally used in many engineering projects.

.

BIBLIOGRAPHY

[ABC+16] Mart$\$'$\$*inAbadi*, *PaulBarham*, *JianminChen*, *ZhifengChen*, *AndyDavis*, *Jeffrey Dean*, −283, 2016.

[ACP21] Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys (CSUR)*, 54(5):1–39, 2021.

[AFK+22] Ahmed Awad, Mostafa M Fouda, Marwa M Khashaba, Ehab R Mohamed, and Khalid M Hosny. Utilization of mobile edge computing on the internet of medical things: A survey. *ICT Express*, 2022.

[AIA] Aianybusiness. `https://youtu.be/reUZRyXxUs4`.

[Amaa] Aws api gateway. `https://aws.amazon.com/api-gateway/`.

[Amab] Aws lambda - serverless compute. `https://aws.amazon.com/lambda`.

[And15] Charles Anderson. Docker [software engineering]. *Ieee Software*, 32(3):102–c3, 2015.

[APP20] Giuseppe Aceto, Valerio Persico, and Antonio Pescapé. Industry 4.0 and health: Internet of things, big data, and cloud computing for healthcare 4.0. *Journal of Industrial Information Integration*, 18:100129, 2020.

[APW+99] Charu C. Aggarwal, Cecilia M. Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast algorithms for projected clustering. In *SIGMOD Conference*, 1999.

[APYS20] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. Batch: Machine learning inference serving on serverless platforms with adaptive batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[art] artillery. `https://www.artillery.io/`.

[ASCG+20] Ricardo S Alonso, Inés Sittón-Candanedo, Óscar García, Javier Prieto, and Sara Rodríguez-González. An intelligent edge-iot platform for monitoring livestock and crops in a dairy farming scenario. *Ad Hoc Networks*, 98:102047, 2020.

[AWS] Awsml. `https://aws.amazon.com/machine-learning/`.

[BAS+21] Praphulla MS Bhawsar, Mustapha Abubakar, Marjanka K Schmidt, Nicola J Camp, Melissa H Cessna, Máire A Duggan, Montserrat García-Closas, and Jonas S Almeida. Browser-based data annotation, active learning, and real-time distribution of artificial intelligence models: from tumor tissue microarrays to covid-19 radiology. *Journal of Pathology Informatics*, 12(1):38, 2021.

[Bat19] Dhaya Sindhu Battina. An intelligent devops platform research and design based on machine learning. *training*, 6(3), 2019.

[BCC+17] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. In *Research advances in cloud computing*, pages 1–20. Springer, 2017.

[BCFPR21] Daniel Belanche, Luis V Casaló, Carlos Flavián, and Alfredo Pérez-Rueda. The role of customers in the gig economy: how perceptions of working conditions and service quality influence the use and recommendation of food delivery services. *Service Business*, 15(1):45–75, 2021.

[Bee17] Jöran Beel. Towards effective research-paper recommender systems and user modeling based on mind maps. *CoRR*, abs/1703.09109, 2017.

[BGLB16] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, Nov 2016.

[BLGN13] Joeran Beel, Stefan Langer, Marcel Genzmehr, and Andreas Nürnberger. Persistence in recommender systems: giving the same recommendations to the same users multiple times. In *International Conference on Theory and Practice of Digital Libraries*, pages 386–390. Springer, 2013.

[BTK22] Ta Phuong Bac, Minh Ngoc Tran, and YoungHan Kim. Serverless computing approach for deploying machine learning applications in edge layer. In *2022 International Conference on Information Networking (ICOIN)*, pages 396–401. IEEE, 2022.

[Bur02] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[BVdB08] Toine Bogers and Antal Van den Bosch. Recommending scientific articles using citeulike. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 287–290. Citeseer, 2008.

[CCH19] Arif Cam, Michael Chui, and Bryce Hall. Global ai survey: Ai proves its worth, but few scale impact. 2019.

[CFT+18] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. A case for serverless machine learning. In *Workshop on Systems for ML and Open Source Software at NeurIPS*, volume 2018, 2018.

[CGLL08] Kannan Chandrasekaran, Susan Gauch, Praveen Lakkaraju, and Hiep Phuc Luong. Concept-based document recommendations for citeseer authors. In *International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 83–92. Springer, 2008.

[Cho17] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[CJL+08] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment*, 1(2):1265–1276, 2008.

[CKE+15] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[CROS21] Dheeraj Chahal, Manju Ramesh, Ravi Ojha, and Rekha Singhal. High performance serverless architecture for deep learning workflows. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 790–796. IEEE, 2021.

[CVZA20] Christopher Culley, Supreeta Vijayakumar, Guido Zampieri, and Claudio Angione. A mechanism-aware and multiomic machine-learning pipeline characterizes yeast cell growth. *Proceedings of the National Academy of Sciences*, 117(31):18869–18879, 2020.

[CZYL14] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu. Web service recommendation via exploiting location and qos information. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1913–1924, July 2014.

[DJL] Djl. `https://djl.ai/`.

[dPa] dPanther digital repository system in florida international university. `http://dpanther.fiu.edu`. Accessed: 2019-04-27.

[dSdAG17] Rodrigo Barbosa de Santis, Eduardo Pestana de Aguiar, and Leonardo Goliatt. Predicting material backorders in inventory management using machine learning. In *2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6. IEEE, 2017.

[DWW+22] Keliang Du, Luhan Wang, Xiangming Wen, Yu Liu, Haiwen Niu, and Shaoxin Huang. Ml-sld: A message-level stateless design for cloud-native 5g core network. *Digital Communications and Networks*, 2022.

[ERK+11] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 2011.

[ES20] Peter Elger and Eóin Shanaghy. *AI as a Service: Serverless machine learning with AWS*. Manning Publications, 2020.

[ESTT18] Radwa Elshawi, Sherif Sakr, Domenico Talia, and Paolo Trunfio. Big data systems meet machine learning challenges: Towards big data science as a service. *Big Data Research*, 14:1–11, 2018.

[fla] flask. `https://flask.palletsprojects.com/en/2.2.x/`.

[GKA+20] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 443–462, 2020.

[GMG01] Gary Geisler, David McArthur, and Sarah Giersch. Developing recommendation services for a digital library with uncertain and changing data. In *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '01, New York, NY, USA, 2001. ACM.

[GMGC22] Joan Giner-Miguelez, Abel Gómez, and Jordi Cabot. A domain-specific language for describing machine learning dataset. *arXiv preprint arXiv:2207.02848*, 2022.

[goo] Ai and machine learning products. `https://cloud.google.com/products/ai`.

[GP06] Marco Gori and Augusto Pucci. Research paper recommender systems: A random-walk based approach. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pages 778–781. IEEE, 2006.

[GXG+22] Francisco X Galdos, Sidra Xu, William R Goodyer, Lauren Duan, Yuhsin V Huang, Soah Lee, Han Zhu, Carissa Lee, Nicholas Wei, Daniel Lee, et al. devcellpy is a machine learning-enabled pipeline for automated annotation of complex multilayered single-cell transcriptomic data. *Nature communications*, 13(1):1–20, 2022.

[had] Hadoop. `http://hadoop.apache.org`.

[HCB+19] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[Hes09] Anders Hessellund. *Domain-specific multimodeling*. IT University of Copenhagen, Innovative Communication, 2009.

[HLL+19] Meng Hao, Hongwei Li, Xizhao Luo, Guowen Xu, Haomiao Yang, and Sen Liu. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics*, 16(10):6532–6542, 2019.

[HZG+21] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Maximilian Augustin, Michael Backes, and Mario Fritz. Mlcapsule: Guarded offline deployment of machine learning as a service. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3300–3309, 2021.

[IAS22] Gerald K Ijemaru, Li Minn Ang, and Kah Phooi Seng. Transformation from iot to iov for waste management in smart cities. *Journal of Network and Computer Applications*, 204:103393, 2022.

[IBM21] IBM. How industry 4.0 technologies are changing manufacturing, 2021.

[Ins21] Fortune Business Insights. Machine learning (ml) market size, share, covid-19 impact analysis. `https://www.fortunebusinessinsights.com/machine-learning-market-102226`, 2021.

[JCXL21] Jananie Jarachanthan, Li Chen, Fei Xu, and Bo Li. Amps-inf: Automatic model partitioning for serverless inference with cost efficiency. In *50th International Conference on Parallel Processing*, pages 1–12, 2021.

[JLH+21] Jingyan Jiang, Ziyue Luo, Chenghao Hu, Zhaoliang He, Zhi Wang, Shutao Xia, and Chuan Wu. Joint model and data adaptation for cloud inference serving. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 279–289. IEEE, 2021.

[JPV+17] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 symposium on cloud computing*, pages 445–451, 2017.

[JW02] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002.

[key] Keystoneml. `http://keystone-ml.org/`.

[KGS+21] Xiangjie Kong, Haoran Gao, Guojiang Shen, Gaohui Duan, and Sajal K Das. Fedvcp: A federated-learning-based cooperative positioning scheme for social internet of vehicles. *IEEE Transactions on Computational Social Systems*, 2021.

[KKS+19] Andreas Kunft, Asterios Katsifodimos, Sebastian Schelter, Sebastian Breß, Tilmann Rabl, and Volker Markl. An intermediate representation for optimizing machine learning pipelines. *Proceedings of the VLDB Endowment*, 12(11):1553–1567, 2019.

[KKZ09] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1, 2009.

[KT00] Steven Kelly and Juha-Pekka Tolvanen. Visual domain-specific modelling: Benefits and experiences of using metacase tools. In *International Workshop on Model Engineering, at ECOOP*, volume 2000, pages 1–9. Citeseer, 2000.

[LDGS11] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[LHM+14] Haifeng Liu, Zheng Hu, Ahmad Mian, Hui Tian, and Xuzhen Zhu. A new user similarity model to improve the accuracy of collaborative filtering. *Knowledge-Based Systems*, 2014.

[LPWG14] Béatrice Lamche, Enrico Pollok, Wolfgang Wörndl, and Georg Groh. Evaluating the effectiveness of stereotype user models for recommendations on mobile devices. In *UMAP Workshops*, 2014.

[LSM14] Thomas Lüke, Philipp Schaer, and Philipp Mayr. A framework for specific term recommendation systems. *CoRR*, abs/1407.1539, 2014.

[LZX+08] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 34. ACM, 2008.

[MD22] Kunal Mahajan and Rumit Desai. Serving distributed inference deep learning models in serverless computing. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 109–111. IEEE, 2022.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[mic] Microsoft azure container service. `https://azure.microsoft.com/en-us/services/container-service/`.

[MIW22] Asif Mahmood, Ahmad Irfan, and Jin-Liang Wang. Machine learning and molecular dynamics simulation-assisted evolutionary design and discovery pipeline to screen efficient small molecule acceptors for ptb7-th-based organic solar cells with over 15% efficiency. *Journal of Materials Chemistry A*, 10(8):4170–4180, 2022.

[MLB+21] Meghana Madhyastha, Kunal Lillaney, James Browne, Joshua T Vogelstein, and Randal Burns. Blockset (block-aligned serialized trees) reducing inference latency for tree ensemble deployment. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1170–1179, 2021.

[MLLI21] Antti Martikkala, Andrei Lobov, Minna Lanz, and Iñigo Flores Ituarte. Towards the interoperability of iot platforms: A case study for data collection and data storage. *IFAC-PapersOnLine*, 54(1):1138–1143, 2021.

[mln] mlnet. `https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet`.

[MNL+10] Cataldo Musto, Fedelucio Narducci, Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Integrating a Content-Based Recommender System into Digital Libraries for Cultural Heritage*, pages 27–38. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[NAC+22] Sudarshan Nandy, Mainak Adhikari, Supriya Chakraborty, Ahmed Alkhayyat, and Neeraj Kumar. Ibonn: Intelligent agent-based internet of medical things framework for detecting brain response from electroencephalography signal using bag-of-neural network. *Future Generation Computer Systems*, 130:241–252, 2022.

[NRMB21] Diana M Naranjo, Sebastián Risco, Germán Moltó, and Ignacio Blanquer. A serverless gateway for event-driven machine learning inference in multiple clouds. *Concurrency and Computation: Practice and Experience*, page e6728, 2021.

[NSRK14] Eirini Ntoutsi, Kostas Stefanidis, Katharina Rausch, and Hans-Peter Kriegel. Strength lies in differences: Diversifying friends for recommendations through subspace clustering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 729–738. ACM, 2014.

[NVI] Nvidia docker. `https://github.com/NVIDIA/nvidia-docker`.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110, 2008.

[PDGQ05] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PMD21] Jean Bertin Nkamla Penka, Said Mahmoudi, and Olivier Debauche. A new kappa architecture for iot data management in smart farming. *Procedia Computer Science*, 191:17–24, 2021.

[PSJ14] Simon Philip, P.B. Shola, and Abari Ovye John. Application of content-based approach in research paper recommendation system for a digital library. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 5(10), 2014.

[PTAJ21] Jason Posner, Lewis Tseng, Moayad Aloqaily, and Yaser Jararweh. Federated learning in vehicular networks: opportunities and solutions. *IEEE Network*, 35(2):152–159, 2021.

[PUL20] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D Lawrence. Challenges in deploying machine learning: a survey of case studies. *ACM Computing Surveys (CSUR)*, 2020.

[RDD19] Partha Pratim Ray, Dinesh Dash, and Debashis De. Internet of things-based real-time model study on e-healthcare: Device, message service and dew computing. *Computer Networks*, 149:226–239, 2019.

[Ric79] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329 – 354, 1979.

[RSR+21] Darius Roman, Saurabh Saxena, Valentin Robu, Michael Pecht, and David Flynn. Machine learning pipeline for battery state-of-health estimation. *Nature Machine Intelligence*, 3(5):447–456, 2021.

[RV96] Raimundo Real and Juan M Vargas. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, 1996.

[SAM15] Sheikh Mohammed Sohan, Craig Anslow, and Frank Maurer. Spyrest: Automated restful api documentation using an http proxy server (n). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 271–276. IEEE, 2015.

[SAS21] Marc Sánchez-Artigas and Pablo Gimeno Sarroca. Experience paper: Towards enhancing cost efficiency in serverless machine learning training. In *Proceedings of the 22nd International Middleware Conference*, pages 210–222, 2021.

[SC05] Alan F. Smeaton and Jamie Callan. Personalisation and recommender systems in digital libraries. *International Journal on Digital Libraries*, 5(4):299–308, Aug 2005.

[SLB+11] Arvind Sujeeth, HyoukJoong Lee, Kevin Brown, Tiark Rompf, Hassan Chafi, Michael Wu, Anand Atreya, Martin Odersky, and Kunle Olukotun. Optiml: an implicitly parallel domain-specific language for machine learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 609–616, 2011.

[SMKL16] Dalia Sulieman, Maria Malek, Hubert Kadima, and Dominique Laurent. Toward social-semantic recommender systems. *Int. J. Inf. Syst. Soc. Chang.*, 7(1):1–30, January 2016.

[SNP19] Roxana Gabriela Stan, Catalin Negru, and Florin Pop. Cloudwave: Content gathering network with flying clouds. *Future Generation Computer Systems*, 98:474–486, 2019.

[SOB10] Sumana Sharma and Kweku-Muata Osei-Bryson. Toward an integrated knowledge discovery and data mining process model. *The Knowledge Engineering Review*, 25(1):49–67, 2010.

[SS22] Raj Mani Shukla and Shamik Sengupta. A novel machine learning pipeline to detect malicious anomalies for the internet of things. *Internet of Things*, 20:100603, 2022.

[Tet22] Laurens Martin Tetzlaff. Bpmn4sml: A bpmn extension for serverless machine learning. technology independent and interoperable modeling of machine learning workflows and their serverless deployment orchestration. *arXiv preprint arXiv:2208.02030*, 2022.

[TK16] Juha-Pekka Tolvanen and Steven Kelly. Model-driven development challenges and solutions: Experiences with domain-specific modelling in industry. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 711–719. IEEE, 2016.

[VKP22] R Vishnu and N Krishna Prakash. Mobile application-based virtual assistant using deep learning. In *Soft Computing and Signal Processing*, pages 609–617. Springer, 2022.

[VVMH20] Edward Verenich, Alvaro Velasquez, MG Sarwar Murshed, and Faraz Hussain. {FlexServe}: Deployment of {PyTorch} models as flexible {REST} endpoints. In *2020 USENIX Conference on Operational Machine Learning (OpML 20)*, 2020.

[WCR+11] Markus Weimer, Tyson Condie, Raghu Ramakrishnan, et al. Machine learning in scalops, a higher order cloud computing language. In *NIPS 2011 Workshop on parallel and large-scale machine learning (BigLearn)*, volume 9, pages 389–396. Citeseer, 2011.

[WDD+19] Kiri L Wagstaff, Gary Doran, Ashley Davies, Saadat Anwar, Srija Chakraborty, Marissa Cameron, Ingrid Daubar, and Cynthia Phillips. Enabling onboard detection of events of scientific interest for the europa clipper spacecraft. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, pages 2191–2201, 2019.

[WKM10] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. Naïve bayes. *Encyclopedia of machine learning*, 15:713–714, 2010.

[WUG+20] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. Dbpal: A fully pluggable nl2sql training pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2347–2361, 2020.

[WWG+18] Wei Wang, Sheng Wang, Jinyang Gao, Meihui Zhang, Gang Chen, Teck Khim Ng, and Beng Chin Ooi. Rafiki: Machine learning as an analytics service system. *arXiv preprint arXiv:1804.06087*, 2018.

[WWGV22] Ying Wu, Yanpeng Wu, Josep M Guerrero, and Juan C Vasquez. Decentralized transactive energy community in edge grid with positive buildings and interactive electric vehicles. *International Journal of Electrical Power & Energy Systems*, 135:107510, 2022.

[WWL+21] Zekun Wang, Pengwei Wang, Peter C Louis, Lee E Wheless, and Yuankai Huo. Wearmask: Fast in-browser face mask detection with serverless edge computing for covid-19. *arXiv preprint arXiv:2101.00784*, 2021.

[xce] xception training data set. `https://github.com/chenshunpeng/Animal-recognition-based-on-xception`.

[ZCD+12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[ZCD+18] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[ZWH+21] Zijian Zhang, Shuai Wang, Yuncong Hong, Liangkai Zhou, and Qi Hao. Distributed dynamic map fusion via federated learning for intelligent networked vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 953–959. IEEE, 2021.

[ZWZ+21] Miao Zhang, Fangxin Wang, Yifei Zhu, Jiangchuan Liu, and Zhi Wang. Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 80–93, 2021.

[ZYY+20] Chengliang Zhang, Minchen Yu, Feng Yan, et al. Enabling cost-effective, slo-aware machine learning inference serving on public cloud. *IEEE Transactions on Cloud Computing*, 2020.

[ZZL+22] Miao Zhang, Yifei Zhu, Jiangchuan Liu, Feng Wang, and Fangxin Wang. Charmseeker: Automated pipeline configuration for serverless video processing. *IEEE/ACM Transactions on Networking*, 2022.

[ZZY+08] Ding Zhou, Shenghuo Zhu, Kai Yu, Xiaodan Song, Belle L. Tseng, Hongyuan Zha, and C. Lee Giles. Learning multiple graphs for document recommendations. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 141–150, New York, NY, USA, 2008. ACM.

VITA

BOYUAN GUAN

Born, Beijing, China

2002-2006	B.A., Software Engineer Beihang University. Beijing, China
2007-2019	M.S., Transportation Engineer Florida International University Miami, Florida
2016-2022	Doctoral Canditdae Florida International University Miami, Florida

PUBLICATIONS

1. (under review) Boyuan Guan, Cheng-Wei Ching, Hailu Xu, Hailu Xu, Yinzhe Zhang, Liting Hu, "Straightline: From-development-to-deployment Multiple Resources-aware Machine Learning Application Pipeline", *In Proceedings of the 2022 Future Generation Computer Systems (FGCS), July 2023*.
2. Boyuan Guan, Liting Hu, Pinchao Liu, Hailu Xu, Jennifer Fu, QingyangWang, dpSmart: A Flexible Group Based Recommendation Framework for Digital Repository Systems", *In Proceedings of the 2019 International Congress on Big Data (Big Data Congress), July 2019.*
3. Hailu Xu, Boyuan Guan, Pinchao Liu, William Escudero, and Liting Hu. "Harnessing the Nature of Spam in Scalable Online Social Spam Detection", *2018 IEEE Big Data workshop on Big Social Media Data Management and Analysis, in conjunction with IEEE Big Data, 2018.*
4. Hailu Xu, Liting Hu, Pinchao Liu, and Boyuan Guan, Exploiting the Spam Correlations in Scalable Online Social Spam Detection", *In Proceedings of the 2019 International Conference on Cloud Computing (CLOUD), June 2019.*
5. Pinchao Liu, Adnan Maruf, Farzana Beente Yusuf, Labiba Jahan, Hailu Xu, Boyuan Guan, Liting Hu, and Sitharama S. Iyengar, Towards Adaptive Replication for Hot/Cold Blocks in HDFS using MemCached", *In Proceedings of 2019 International Conference on Data Intelligence and Security (ICDIS 2019), June 2019.*
6. Huang, Jian, Yanni Wang, Zheli Liu, Boyuan Guan, Dan Long, Xiaoping Du "On modeling microscopic vehicle fuel consumption using radial basis function neural network." *Soft Computing 20.7 (2016): 2771-2779.*

7. Bakker,Rebecca Jean, Boyuan Guan, Liting Hu, Kelley Rowan “AI and the Archive: Testing Facial Recognition in Digital Collections” Fund Project, *Lyrasis 2020*.