American Journal of Applied Sciences

Literature Reviews

Machine Learning Software Architecture and Model Workflow. A Case of Django REST Framework

[1*]Kennedy Ochilo Hadullo and [2]Daniel Makini Getuno

[1]Institute of Computing and Informatics, Technical University of Mombasa, Mombasa, Kenya
[2]School of Education, Department of e-learning, Egerton University, Njoro, Kenya

Article history
Received: 15-02-2021
Revised: 24-05-2021
Accepted: 04-06-2021

Corresponding Author:
Kennedy Ochilo Hadullo
(Technical University of Mombasa) Institute of Computing and Informatics, Mombasa, Kenya
Email: khadullo@gmail.com

Abstract: The purpose of this study was to find out the challenges facing Machine Learning (ML) software development and create a design architecture and a workflow for successful deployment. Despite the promise in ML technology, more than 80% of ML software projects never make it to production. As a result, majority of companies around the world with investments in ML software are making significant losses. Current studies show that data scientists and software engineers are concerned by the challenges involved in these systems such as: limited qualified and experienced ML software experts, lack of collaboration between experts from the two domains, lack of published literature in ML software development using established platforms such as Django Rest Framework, as well as existence of cloud software tools that are difficult use. Several attempts have been made to address these issues such as: Coming up with new software models and architectures, frameworks and design patterns. However, with the lack of a clear breakthrough in overcoming the challenges, this study proposes to investigate further into the conundrum with the view of proposing an ML software design architecture and a development workflow. In the end, the study gives a conclusion on how the remedies provided helps to meet the objectives of study.

Keywords: Machine Learning, Data Science, Software Engineering, Development, Deployment, Django REST Framework, Architecture, Workflow

Introduction

Artificial Intelligence(AI) has become an important area of research in the 21st Century in many fields including: Marketing, education, banking, finance, agriculture, healthcare, space exploration, autonomous vehicles, law and so forth (Keshari, 2020; Hull, 2020). Besides, AI has also long been a major focus for tech leaders such as: Facebook, Amazon, Microsoft, Google and Apple (FAMGA) who have all been aggressively acquiring AI startups by trying to integrate machine learning into their products and services (Pathak, 2017).

It is noteworthy that FAMGA have announced shifting from a mobile-first world to an AI-first world (Allad, 2016). The shift implies that Information and Communication Technology (ICT) focus has moved from optimizing user experience through mobile phone inter faces to maximizing predictive accuracy through the use of AI.

The AI domain consists of several subfields, such as Machine Learning (ML), Deep Learning (DL), natural language processing, image processing and data mining which are also important topics in computing research and technology industries (Zhang and Tsai, 2005; Zhang *et al*., 2019). ML is an application of AI that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed.

Despite the interest caused by ML due to its wide applications and benefits in computing technology, DL, a subfield of machine learning is attracting much attention as well. DL uses artificial neural networks to mimic the workings of the human brain in processing data and creating patterns for use in decision making (Zang *et al*., 2019). However, despite the potential created by both ML and DL in data science projects, there is evidence that majority of the projects do not make it to production (Redapt Marketing, 2019; Ameisen, 2020) with a high failure rate of approximately up to 90% being reported.

DOI: 10.3844/ajassp.2021.152.164

Motivation

Although significant strides have been made in the field of ML software development, there is still a considerable amount of pitfalls that slow down the application of the technology in investments worldwide. Some of the issues that have motivated this study are stated below.

Most papers on ML published by students undertaking masters and PhD studies in computer science and Information Technology have been implemented using Python programming language and deployed using Tkinter (Grayson, 2000), graphical user interface for python desktop application.

Secondly, the use of REST frameworks (Django or Flask) for Machine learning web applications is quite complex and requires a good architecture and a clear workflow which are currently lacking (Jordon, 2019).

Thirdly, there is generally lack of clear software engineering principles for successfully integrating ML models and web applications.

Finally, the reportedly high failure rate of ML software projects of up to 80% calls for further research into how the design, development and deployment maybe be enhanced to improve the ML software engineering.

Background

Despite the perceived benefits of ML applications, the process of developing, deploying and continuously improving them is more complex compared to the traditional software, such as a web services and mobile applications (Geron, 2019; Chen, 2015). Deployment, or simply, putting models into production implies making it available to others, whether that be users, management, or other systems. When successfully deployed, ML projects enables users to send data and get their predictions accurately via web or mobile interfaces.

During the development of an ML software, there are three major tasks undertaken by the developers: The creation of ML model, the design of web application for running the model and the successful deployment of the product as an intelligent software (Chen *et al*., 2020; Li *et al*., 2015; Washizaki *et al*., 2019). These tasks are quite complex and demanding and require the relevant skills and inputs of from both ML Engineers and Software Engineers.

Task one requires a thorough knowledge of ML modeling using a machine learning programing language such as python. Task two requires the knowledge of web development using a REST framework and the integration of the model with the web application. Finally, task three involves successful deployment of the application with reliable outputs.

The challenges faced by ML engineers have resulted into more research being conducted in this area with the view of alleviating the challenges mentioned. As a result, new software design patterns and new platforms for development have emerged (Ameisen, 2020; (Zhang and Tsai, 2005; Zhang *et al*., 2019; Geron, 2019). However, these platforms both advantages and disadvantages.

The advantages include: Better data visualization, scalability, pipelining and code debugging options. On the flipside, the use of these tools requires fundamental knowledge of advanced calculus and linear algebra along with a good understanding of web based software engineering in order to create a sustainable ML software.

Secondly, the field of data science is known to mainly focus on ML algorithm writing and model development using data mining software's such as WEKA, Rapid Mining and Orange (Mikut and Reischl, 2011) and or ML programming languages such as python and R (Moroney, 2020), with preferably labeled data, having minimal dimensionality and optimizing performance and accuracy of the model (Schröer *et al*., 2021).

Another cause of concern in ML software development is that the principles used in software engineering and ML modeling are quite divergent: While ML is concerned more with algorithm writing, testing and accuracy issues, software engineering deals mainly with scalability, extensibility, configuration, consistency, modularity and security issues etc. (Sculley *et al*., 2015). It is thus difficult to produce a software that seamlessly combines constraints from both domains.

Lastly, there is no clear formula or procedure on the integration of ML models with web applications created with Django or Flask. This is to imply that while a majority of data scientists are good at creating ML models using datamining tools, very few are good at creating the same models using languages such as R or Python.

The problem is further compounded by the need to design and develop a web application and merge it with an ML application as one application (Plonski, 2019; Bajpai, 2020).

Purpose

The purpose of the study was to Identify the challenges that hinder the Development and Deployment of Machine Learning Software Models and thereafter create a Software Architecture and a Deployment Workflow implementable using Pythons Django Rest Framework (DRF).

Study Objectives

Identify the challenges facing data scientists and software engineers during Machine Learning Software Development and Deployment:

i) Develop a suitable Machine Learning Software Architecture that is deployable with Python's DRF

ii) Integrate a Machine Learning Software Deployment Workflow based on the software architecture created in objective (i).

In order to answer the study objectives, we propose to come up with a Software Design Architecture (SDA) to better understand the basic structure of a ML software and a Software Deployment Workflow (SDW) to guide the development and deployment of ML software and help overcome the challenges identified in the study.

Literature Review

The study reviewed the literature relevant to the study by using the Framework by proposed by Murad (2020) and illustrated in Fig. 1. By applying this framework, we decided to use a systematic literature review and scoped the existing literature on ML software to help us define the Research Problem (RP). Once this was done, the RP was specified in a clear and structured manner by framing it using specific keywords.

Some of the keywords used included machine learning software development, machine learning software deployment, machine learning engineering, machine learning web applications, data science engineering, machine learning software architecture, machine learning software workflow, Django REST framework and the challenges of deploying machine learning models.

To capture as many relevant articles as possible, a range of journals, books and grey literature in the mentioned areas were searched extensively to identify whether they contained articles having these key words. In total, twenty-five journals (25), sixteen books (16) and thirteen (13) grey literature were scoped. Out of these, only 18 journals, 15 books and 8 grey literature were found to be relevant for review

Some of journals included were: Journal of Systems and Software, SSRN Electronic Journal, International Journal for Research in Applied Science and Engineering Technology, Journal of Data Warehousing and Journal of Systems, Software and Willy online Library. The review enabled us to identify some of the processes, models, frameworks and related work within the scope of the study topic as described in the next sections.

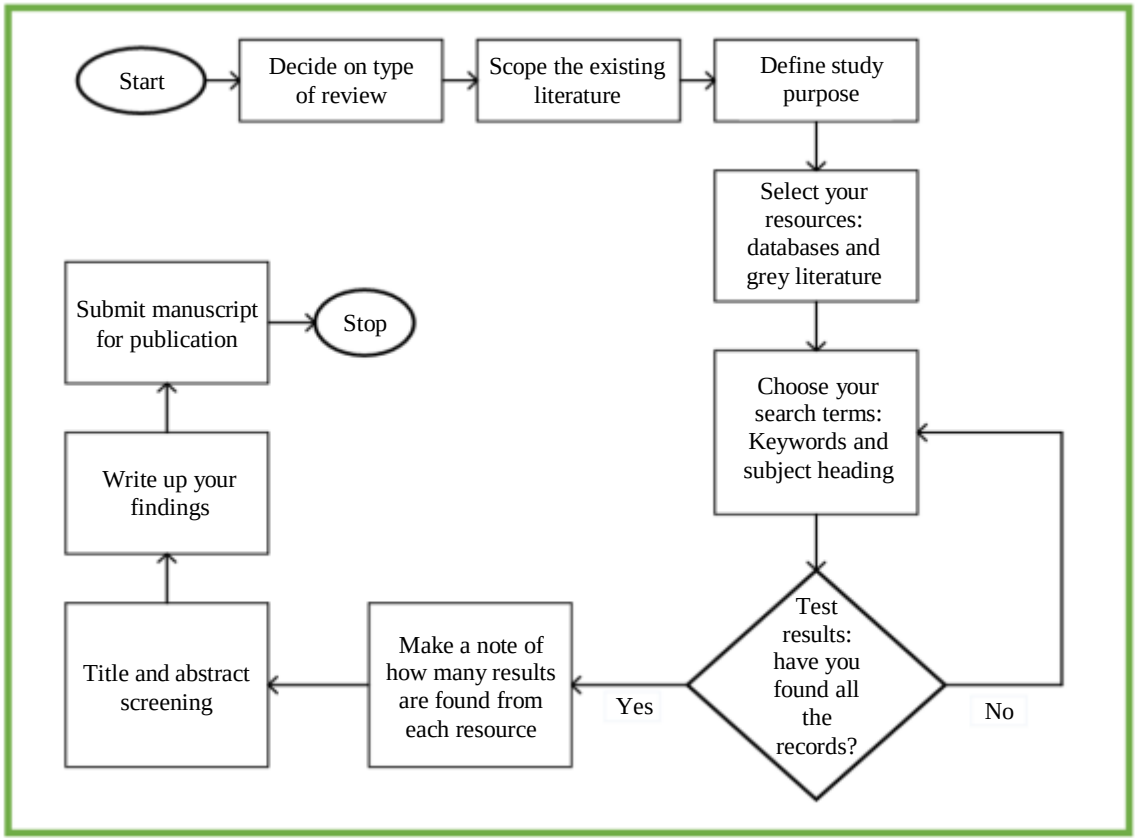

Fig. 1: Literature review flowchart (Murad, 2020)

Machine Learning as a Model (MLaaM)

MLaaM is the output of writing ML algorithms that run on data and represents what was learned by the algorithm on training data. An algorithm in ML is a procedure that is run on data to create a machine learning model. Examples of ML algorithms include: K-nearest neighbors for classification, linear regression for regression and k-means for clustering (McClendon and Meghanathan, 2015).

The model is a file that is saved after running the algorithm and represents the data, the rules and the procedures for using the data to make a prediction (Geron, 2019). The most popular programming language for MLaaM is Python while Tensor Flow (TS) is the most preferred software framework by developers for both DL and ML (Jaxenter, 2018).

ML models can be created using three techniques: Supervised learning, unsupervised learning and reinforced learning. Supervised learning algorithms which are the most common are trained using labeled examples, such as an input where the desired output is known, while unsupervised learning is used against data that has no historical labels (Sharma, 2020).

Machine Learning as a Service(MLaaS)

Machine learning as a service (MLaaS) refers to a number of services that offer machine learning tools as part of cloud computing services (Singh, 2021; Geron, 2019). The main benefits of these tools is that customers can get started with machine learning applications quickly without installing specific software or provisioning their own servers. MLaaS providers offer services for the development and deployment of ML software projects that allow: Data transformation, predictive analytics, data visualization and advanced ML algorithms (Geron, 2019; Zhang and Tsai, 2005; Zhang *et al*., 2019; Singh, 2021).

MLaaS providers normally guarantee to their clients all stages of the machine learning process, including data storage and management, model development and deployment, performance monitoring and support and ensuring maximum efficiency of the whole machine learning process (Zhang and Tsai, 2005; Zhang *et al*., 2019).

Different providers may vary slightly in their cloud services, however most of them offer environments that can be used to: Prepare data, train, test, deploy and provide performance monitoring. Some of the popular providers include Amazon Web Services (Bankar, 2018), Google (Sanderson, 2012), IBM (Miller, 2019), Microsoft Azure (Ranjeetsingh, 2014) and Uber (Oppegaard, 2021).

ML Model Software Deployment

Software deployment is all of the activities that make a software system available for use. It is the mechanism through which applications modules are delivered from developers to users. The methods used by developers to build, test and deploy new code will impact how fast a product can respond to changes in customer preferences or requirements and the quality of each change (Fitzgerald and Stol, 2017).

In the context of ML, the process of taking a trained model and making its predictions available to users is known as deployment. As such, ML deployment is not very well understood amongst data scientists who lack backgrounds in software engineering. Alternatively, most software engineers are not good in ML model development. Plonski (2019) highlighted the four methods of deployment, outlining the requirements, merits and the demerits of each. The methods are summarized in Table 1.

Table 1: Different ways of deploying ML models. Adopted from Plonski (2019)

SN	Deployment Method	Requirements	Advantage	Disadvantages	Comment
1	Locally (Laptop or computer)	-Jupiter Notebook - R Studio or -Weka	Simple to implement Predictions on ML code	Hard to govern, monitor, scale and collaborate.	Not recommended for production
2	Hard-code the ML algorithm in the system's code.	-Jupiter Notebook - R studio or -Weka -Software program	Can be used with simple ML algorithms, like Decision Trees or Linear Regression	Hard to govern, monitor, scale and collaborate	Not recommended for production
3	Use of REST API or Web Sockets.	-Jupiter Notebook, -R studio or -Weka. -Software Framework	All requirements for the ML production system can be fulfilled.	Requires Data Scientist and Software Engineer	Recommended for production
4	Use of a commercial cloud vendor for Deployment	Colab PDE or Jupiter Notebook on Laptop	All requirements for the ML production system can be fulfilled.	Requires Data Scientist and Software Engineer	Recommended for production

Django REST Framework (DRF)

Django Representational State Transfer (REST) Framework is a free and open source high-level Python web framework that encourages rapid development and clean, pragmatic design. DRF is a powerful and flexible toolkit used for rapidly building web applications based on Django database models (Jordon, 2019; Bajpai, 2020) with the following advantages: Secure, scalable, customizable application with serialization that supports both the Object Relational Mapping(ORM) and non-ORM data sources (Jordon, 2019; Bajpai, 2020).

Given that most ML models are created using Python programing language makes DRF a preferred platform for ML software development.

ML Model Software Architecture(MMSA)

The ML software application building process is a complex process that brings together several components constituting the software engineering life cycle: Requirement engineering, analysis, design, development, testing deployment and maintenance (McGovern *et al*., 2004).

Thus, there is need for a software architecture that supports the ML model component and the web application components and without negatively affecting the performance of the software (Binge, 2020).

IEEE CS (2000) defines Software Architecture(SA), SA as the fundamental organization of a software embodied in its components, their relationships to each other and the principles guiding its design and evolution.

The SA for this study will consist of the following components: The architectural pattern which defines the granularity of a component, system Interaction which defines how the components communicate with each other and software quality attributes such as: Scalability, extensibility, maintainability, portability, adaptability and resilience, etc.

However, it is important to note that the type of architecture used in a software is normally determined by the project objectives, the proposed budget, the developer team skillset, infrastructure limits and the stakeholders interest (Binge, 2020).

Machine Learning Operations

Machine Learning Operations or "MLOps" is defined as the practice for collaboration between data scientists and software engineers in automatically managing the deployment of ML and DL software lifecycles (Wang, 2019). MLOps can be manual or automatic

The manual MLOps processes as illustrated by Fig. 2 is an entirely manual process that includes data analysis, data preparation, model training and validation in Jupiter Notebook by data scientists. The data scientists hand over a trained model as an artifact to the software engineering team for deployment by putting the trained model in a code repository (Singh, 2021). The software engineers deploy the model as a prediction service using a micro service architecture with REST APIs. The workflow of this process is illustrated in Fig. 2.

Related Work

A study by Runyu (2020) to create a design pattern for ML deployment ascertained that although data scientists have come up with many good algorithms and trained models, putting those models is still a challenge. The key obstacles hindering ML software production are: Lack of a clear methodology for moving ML models to production, use of monolithic programming or lack of modularization when writing ML code and obscure best practices in ML software development.

Runyu (2020) developed a system design pattern named Model-Service-Client + Retraining (MSC/R) in order to overcome these challenges (Fig. 3). This design pattern incorporates the principles of modularization and separation of concerns and uses a micro service RESTful API architecture. Figure 3 illustrates the architecture.

The MSC/R design pattern works by using three teams of distinct developers: Data scientists-working on the model, MLOps engineers-working on the service and client developers-working on the front end. Then the next important part of the design illustrates connectors linking the four main system components: Model, service, retraining and client. The connectors main function is to provide guidelines for collaborations between the system components during development.

In a related study by O'Leary and Uchida (2020) to identify the common problems with creating ML pipelines from existing code, data was collected via face to face meetings in coding workshop settings averaging 100 companies, data scientists, researchers, ML platform owners and software engineers. The companies interviewed were in the process of transforming their business through the use of ML.

The projects involved migrating existing ML models to MLaaS using Kube Flow Pipelines (KFP) and Tensor Flow Extended (TFX). The study identified three problems: Firstly, due to the highly iterative nature of ML model development, the coding does not usually follow object oriented principles such as modularization and code re-use making it unsuitable for deployment using software engineering principles. As a result, engineers often need to re-implement the model from scratch into a deployable software. During the re-implementation, many of the implicit assumptions made by data scientists for modeling get lost, resulting in unexpected inconsistencies and issues in production.

Secondly, most ML model developments use "monolithic programming approaches" i.e., building applications that are "single-tiered" in nature. Single-tier architecture when used in ML combines data with

business logic and user interface codes in a single logical structure. This results into a tightly coupled application that becomes inefficient to run and difficult to maintain.

In a related study by Sculley *et al*. (2015) that set out to explore the several specific design risk factors to account for in ML software deployment, the output was the Technical Debt Framework (TDF) illustrated in Fig. 4. Technical debt is an analogy used to describe a situation in software development where a workaround is used to solve a software problem (Kruchten *et al*., 2012; Zazworka *et al*., 2011). Several technical problems (debts) and potential workarounds (repayment approaches) were identified and used to create the TDF (Fig. 4).

Default in payment of technical debts may hinder successful deployment. The debts include issues related to: Design, coding, testing, documentation, versioning and infrastructure. Repayment can be done via: Automation, re-writing, refactoring, re-engineering, re-packaging, bug fixing and improving documentation. Repayment results into an improved software quality. ML systems have a tendency for incurring technical debts because of the already stated problems related to the domains of ML and software engineering.

Another study that set out to identify the challenges in deploying DL software by Chen *et al*. (2020), proposed an ML deployment process consisting of four phases: DL data collection, DL model training, Model conversion and exportation to TS and Platform configuration and deployment (Fig. 5).

The DDDM has two facets: DL software development and DL software Deployment. The first facet makes use of TF and Keras to integrate models into software applications for real usage after validation and testing. The second facet involves deploying the model on a cloud based server platform such as AWS SageMaker or Google Cloud.

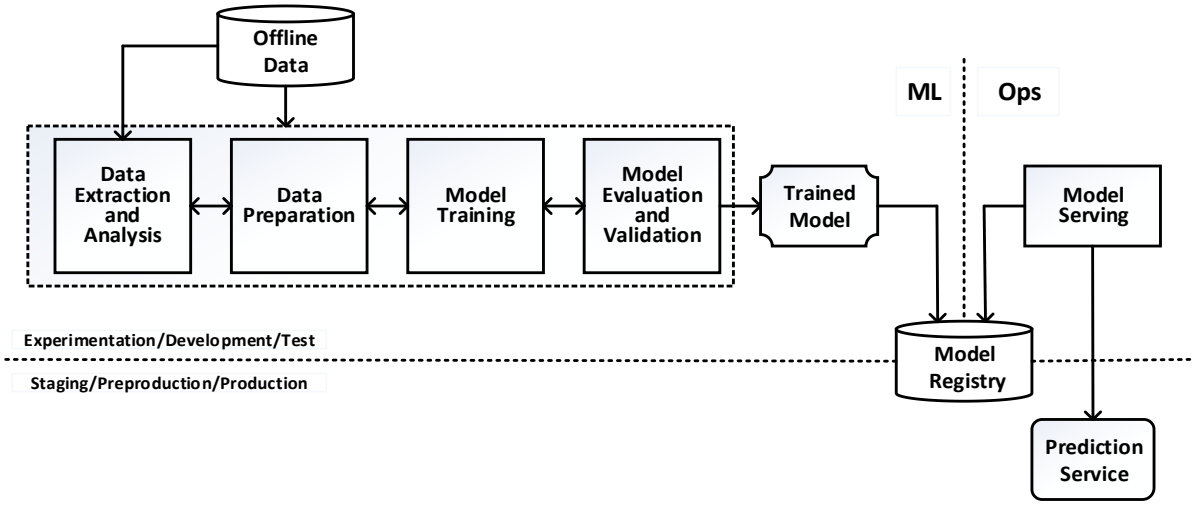

Fig. 2: Manual MLOps deployment process Singh (2020)

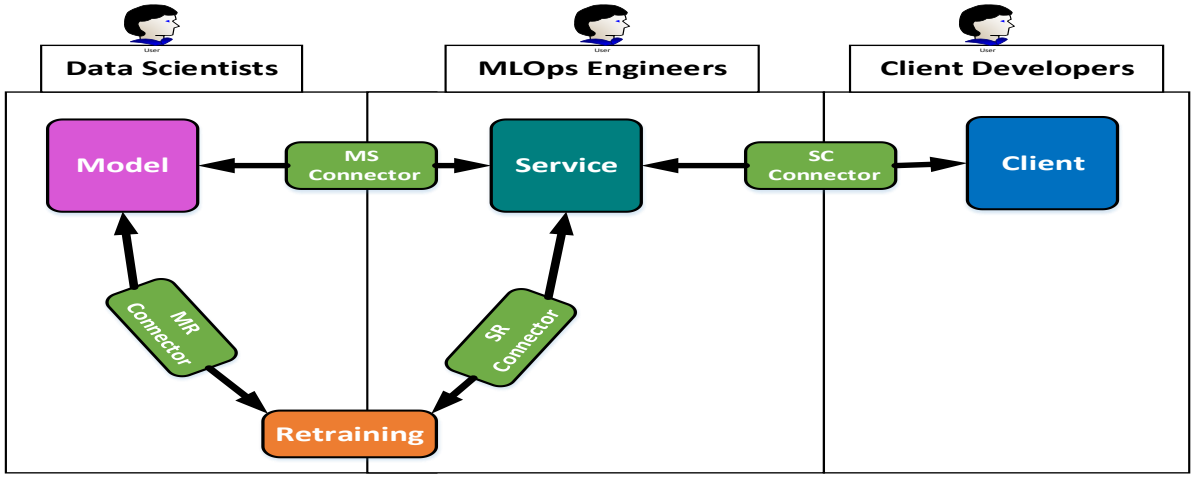

Fig. 3: The Model-Service-Client + Retraining (MSC/R) design pattern. Source (Adopted from Runyu, 2020).

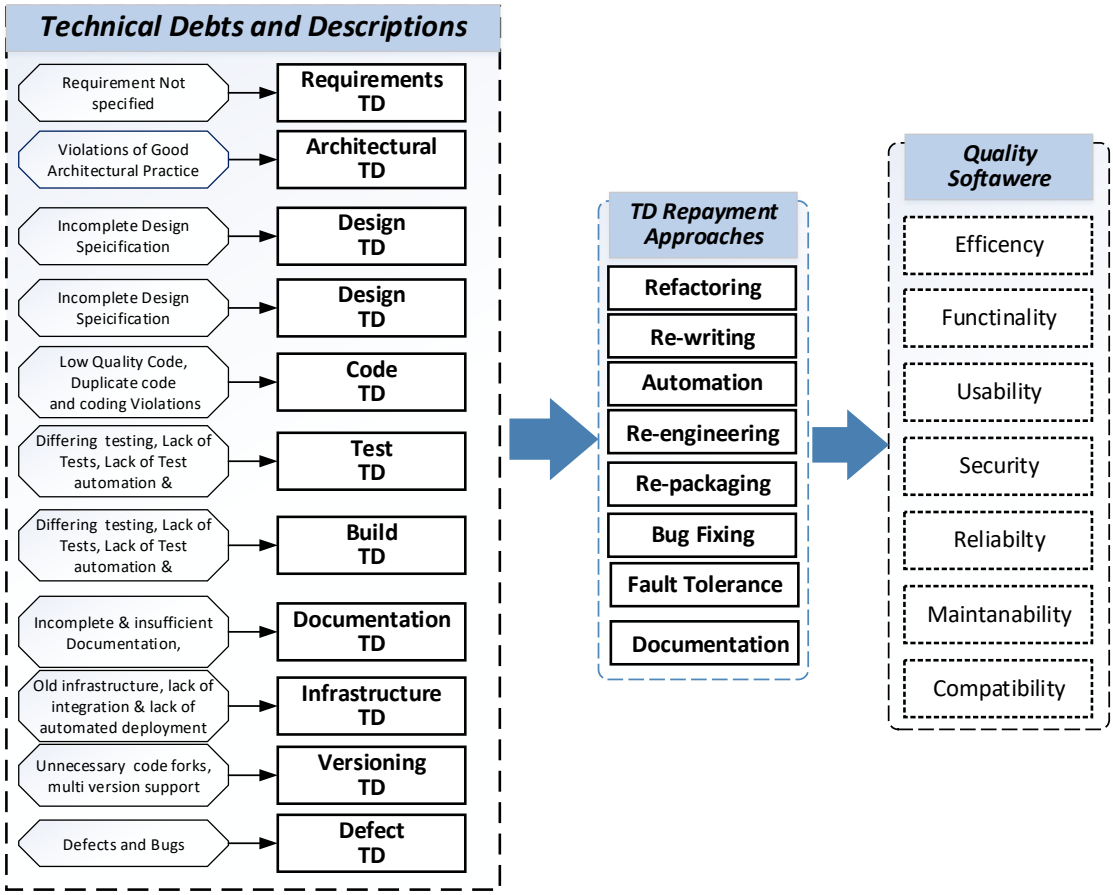

Fig. 4: The technical debt framework source: Adopted from Li *et al*. (2015)

The deployment challenges identified include: Converting models to platform formats, configuration errors encountered during integration, limited skills in ML software development and data processing challenges when converting raw data into the input format needed by the model software. To obtain the data relevant for the study, over 3,023 posts from (Stack Overflow, 2020), specifically from TS serving, Google cloud ML and Amazon sageMaker were collected and analyzed.

In another related study, Esmaeilzadeh (2017), designed an architecture and developed a testable, scalable and efficient web-based application that models and implements machine learning applications in cancer prediction. The main components that formed the architecture of the system included a server, a database, a programming language, Django web framework, front-end design, testability, scalability, performance and design pattern (Fig. 6).

The data set for the study's application was a subset of the Surveillance, Epidemiology and End Results (SEER) Program of the National Cancer Institute. The application was implemented with Python as the back-end programming language, Django as the web framework and MYSQL as the database server. The front layer of the application was built using HTML CSS and JavaScript.

The study used Automated testing approaches to ensure the following: Making sure the application is working as expected before deployment, ensuring that new functionalities do not change the behavior of application in unexpected way, finding and fixing bugs and testing the performance of the application under heavy loads.

Washizaki *et al*. (2019) embarked on a study with the purpose of collecting, classifying and discussing the best practices for designing quality and complex ML systems (Fig. 7).

The study set out to collect good and bad design patterns for ML software so as to provide developers with a comprehensive classification of such patterns. By using a questionnaire-based survey, the study established that there is a lack of expertise by ML engineers on the development of the architectures and design patterns. The study formulated a design pattern based on the Model View Controller (MVC) pattern having three layers: Presentation Layer, the Logic Layer and the Data Layer.

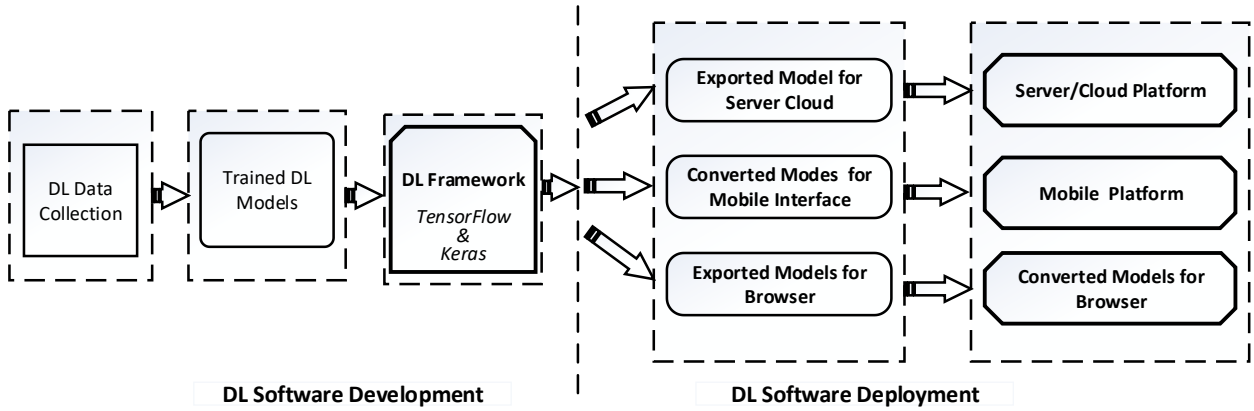

Fig. 5: DL software Development and Deployment Model (DDDM). Source (Adopted form: Chen *et al*., 2020)

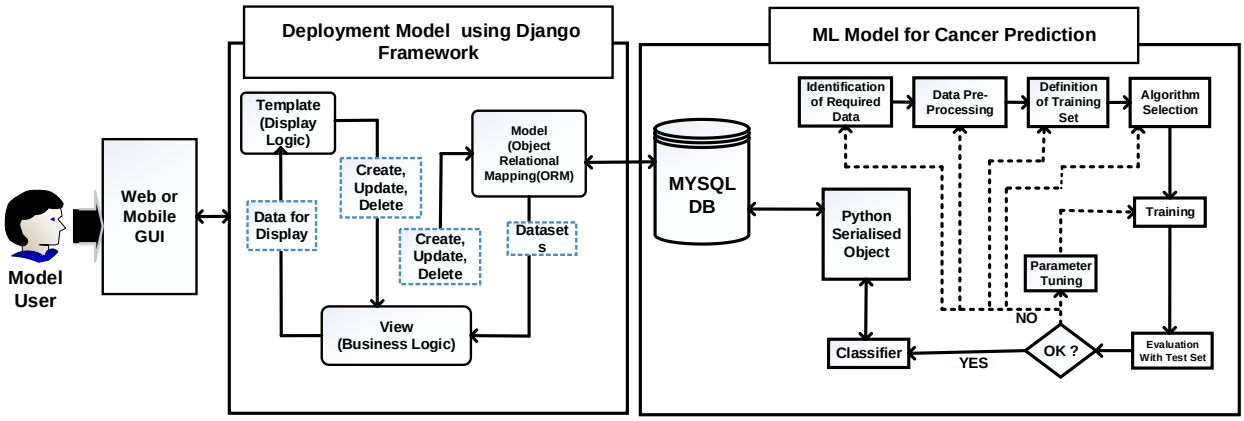

Fig. 6: ML software deployment architecture adopted from Esmaeilzadeh (2017)

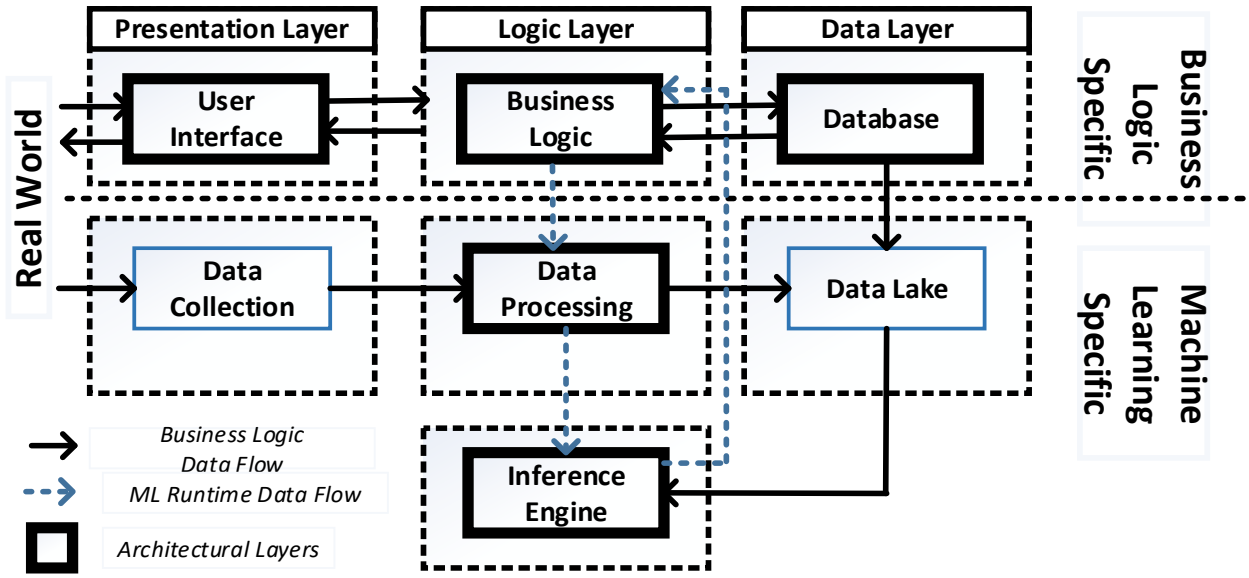

Fig. 7: Software engineering design pattern ML software systems. Adopted from Washizaki *et al*. (2019)

Table 2: Summary of literature review

SN	Model/Pattern/Framework and Author	Advantages	Disadvantages
1	Model-Service-Client + Retraining (MSC/R) design pattern (Runyu, 2020)	This design pattern incorporates the principles of modularization and separation of concerns and uses a micro service	Does not clearly how the model, service and client component are integrated
2	A test driven approach to develop web-based machine learning applications (Esmaeilzadeh,2017)	Use of automated testing approaches to ensure the following: Making sure the application is working as expected before deployment.	No clear explanation how the rest of the application was developed using Python, Django, MYSQL CSS and JavaScript. No mention of how deployment was done
3	DL software deployment model: Chen *et al*. (2020)	Uses tensor flow and Keras to integrate models into software applications deploying the model on a cloud based server platform such as AWS sage maker	No clear methodology on how Tensor Flow and Keras was used No clarity on how deployment was done
3	Software engineering design pattern for designing machine learning Systems (Washizaki *et al*., 2019)	Exposed lack of expertise by ML engineers on ML software development created an MVC for ML software	No clear methodology on how the logic, the data and the presentation layers were created, integrated and deployed together with the ML model
4	The Technical Debt Framework (Adopted from **(**Li *et al*., 2015).	Identified some of the risks in ML software deployment called technical debts. Identified debt repayment approaches.	No mention of an architecture or a deployment workflow
5	Common problems with creating machine learning pipelines from existing code (O'Leary and Uchida, 2020)	Used KubeFlow Pipelines (KFP) and TensorFlow Extended (TFX).	Methodology on both development and Deployment not clear

Summary of Literature Review

After a comprehensive literature review, the results are summarized based on the model or framework reviewed, in terms of the advantages and disadvantages of each framework and model (Table 2).

Proposed ML Software Model Deployment Architecture(DFMSA)

The proposed architecture describes the major components of both the ML model and the Django part, their relationships (structures) and how they interact with each other. This architecture is known as the Django Framework ML Software Architecture(DFMSA). The DFMSA consists of six sub architectures(SAs): The user interface component, the Serialization/De-Serialization component, the server/repository component and the application component, configuration files component and the command line utility component (Fig. 8).

SA1: User Interface

The user interface provides a connection between the Admin and normal user with the system through the Admin Panel and the Client Interface. Beneath this SA lies the static and template folders containing the CSS, HTML, JavaScript and JSON files. The SA connects with the rest of the application through the application URLS file.

SA2: Django API

The Django API is made up of the files: View.py for logic,models.py for database code, apps.py for application configuration, urls.py for providing paths, admin.py for administrative functions and tests.py for writing tests. All the files work in conjunction to make the application accept user data and give predictions.

SA3: System Configuration Files

The configuration files such as the settings.py and urls.py are vital in linking the system files together. For example, they are useful in creating paths and importing.

For example, they are useful in creating paths and importing files, linking the static and template files, defining database credentials and middleware components and linking the installed apps and security key.

SA 4: Serialization/De-Serialization

Object serialization is the process of saving a ML Model as a Pickle, a Joblib or manually saving and restoring using a JSON approach. Serialization represents an object with a stream of bytes, in order to store it on disk, send it over a network or save to a database. Deserialization is the process of restoring and reloading the pickled ML Model back to the Jupiter Notebooks(ipynb) format.

SA 5: Server and Repository

Heroku is a Cloud Platform as a Service (PaaS) supporting several programming languages such as: Ruby, Java, Node.js, Scala, Python and PHP. One advantage with Heroku is that If the project is already pushed to GitHub, automatic deployments can easily be set from the project's repository in GitHub from the Heroku dashboard.

SA 6: Command Line Utility

The command line utility contains two major utilities: Manage.py, a command-line utility that lets you interact with this Django project in various ways and django-admin.py, a Django's command-line utility for administrative tasks.

Proposed ML Software Model Deployment Workflow(SMDW)

The proposed SMDW is arrived based on the proposed architecture and Literature Review summary (Table 2).

From this table, there are six factors which we turned into six phases: Start, build ML Model, Build Django App, integrate Model into Application or App, Make Predictions and test user response tests using two variants: Variant A vs variant B, also known as A/B testing (Fig. 9).

Phase 1: Start

During this phase, the software engineer is supposed to start by setting up a GitHub account, installing the Python virtual environment, creating a Django project and adding applications files into the project followed by committing the code into the GitHub Repository (GHR). This is in preparation for the software engineering part of the project.

Phase 2: Build ML Model

During this phase, an ML engineer or a data scientist installs Jupiter Notebook and installs and loads all the initial packages required for the project. This is followed by the loading and pre-processing of the data file, writing, training and saving the algorithms before adding the code into the GHR.

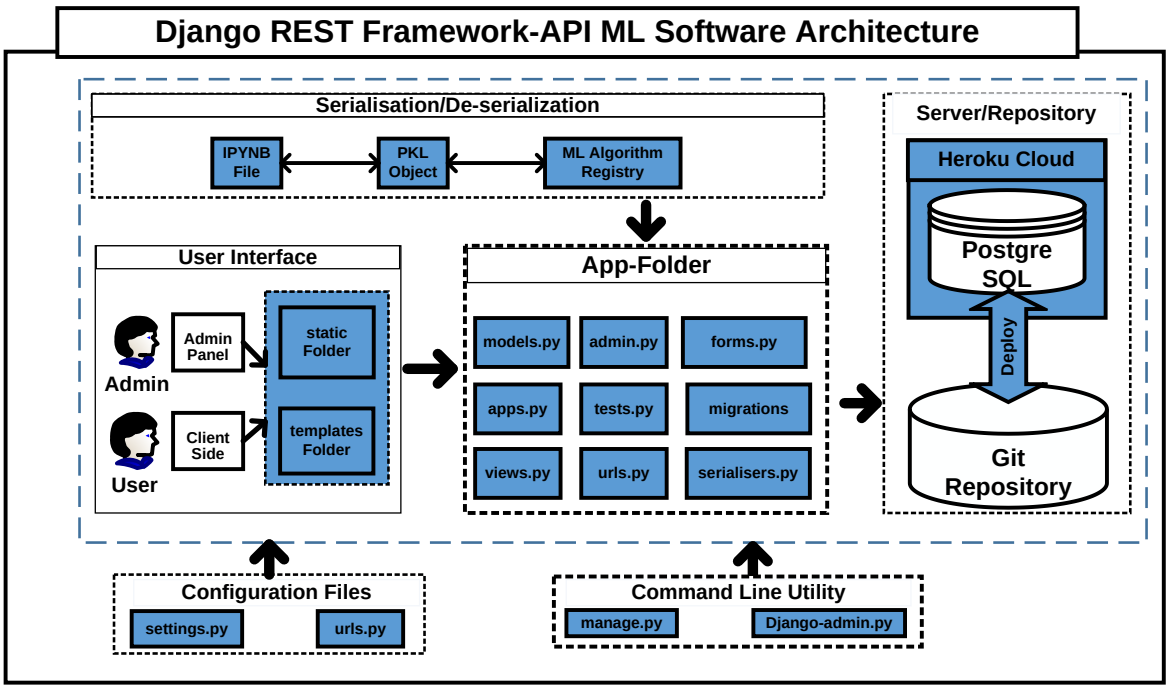

Fig. 8: Proposed system DFMSA using Plonski (2019)

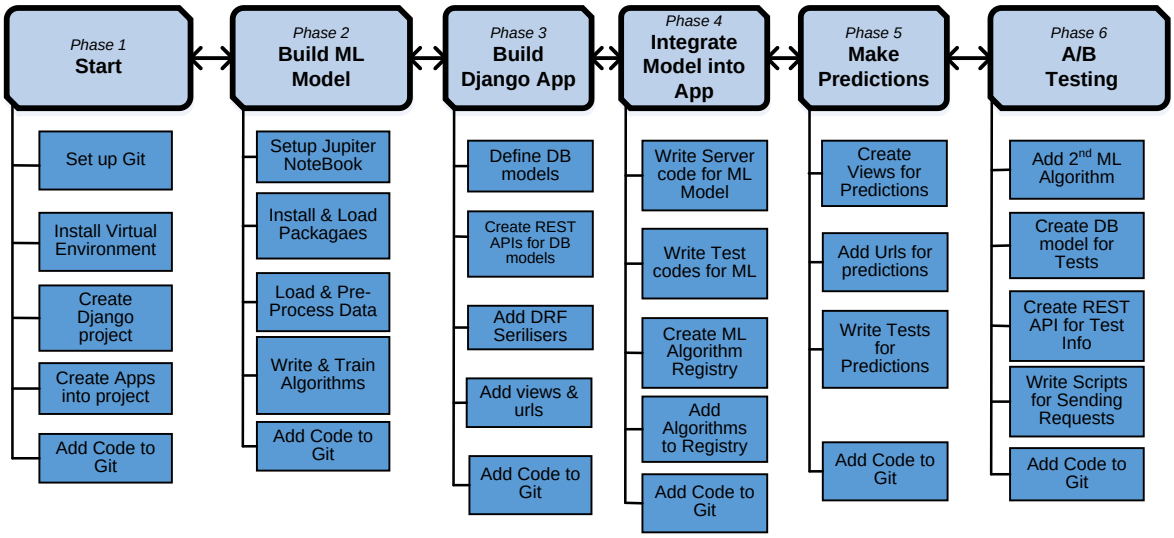

Fig. 9: Proposed ML model deployment workflow Plonski (2019)

Phase 3: Build Django App

During this phase, the software engineer continues with what was started in Phase 1 by adding the database models, creating the REST APIs for the models, adding DRF serializes, adding views and URLS and adding the code into the GHR.

Phase 4: Integrate ML Model in Django App

During this phase, the software engineer continues with what was done in Phase 3 by writing ML server code for the model, write Test codes, creates a registry and add algorithms into the registry and then add the code into the GHR.

Phase 5: Make Predictions

During this phase, the software engineer continues with what was done in Phase 4 by creating views for predictions, creating DB models for Tests, create REST APIS for Tests, write scripts for sending Requests and add the code into the GHR.

Phase 6: A/B Testing

A/B testing in the context of this study is the process of comparing two outputs of the ML software predictions and concluding which of the two outputs or variants is more effective or accurate. The other parts of the project are repeated such as creating views for predictions, creating DB models for Tests, creating REST APIS for Tests, writing scripts for sending Requests and adding the code into the GHR.

Conclusion and Recommendations

This study investigated challenges that hinder the Development and Deployment of ML software models in order to create an architecture and a deployment workflow implementable using Pythons DRF. After a systematic literature review, the main challenges were found to be: Unethical programming practices, lack of software development skills that integrate both data science and software engineering, difficulty in using software's and tools for developing ML software and a lack of clear methodology for deployment. A suitable ML software architecture and model workflow and are also presented as a solution to deployment problems within the ML engineering. This study aims to benefit ML software engineers in industry to help increase the rate of production as well as masters and PhD students in IT and computer science to help them in wring their thesis regarding ML software. It is recommended that there I need to use the created architecture and deployment workflow to try and deploy an ML software as a test.

Acknowledgement

This research was made possible by the support provided by The Technical university of Mombasa and Egerton university through journal subscriptions, need-based acquisition and a favorable research environment.

Author's Contributions

Kennedy Ochilo Hadullo: Contributed mainly on the architecture design of the manuscript.

Daniel Makini Getuno: Contributed mainly on the workflow design of the manuscript.

Ethics

This article is original and contains unpublished material. Both the corresponding author and the co-author confirm that they have read and approved the manuscript and that no ethical issues are involved. The authors declare that they have no competing interests.

References

Allad, R. (2016). Moving from a Mobile First to an AI First World. https://unionstreetmedia.com/moving-from-a-mobile-first-to-an-ai-first-world/

Ameisen, E. (2020). Building Machine Learning Powered Applications: Going from Idea to Product. " O'Reilly Media, Inc.".

Bajpai, S. (2020). Analyzing resume using natural language processing machine learning and django. International Journal for Research in Applied Science and Engineering Technology, 8(5), 2037-2039. https://doi.org/10.22214/ijraset.2020.5333

Bankar, S. (2018). Cloud Computing Using Amazon Web Services AWS. International Journal of Trend in Scientific Research and Development, 2156-2157. https://doi.org/10.31142/ijtsrd14583

Binge, S. (2020). The importance of good software architecture. https://www.sitepen.com/blog/the-importance-of-good-software-architecture

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. IEEE Software, 32(2), 50-54. https://doi.org/10.1109/ms.2015.27

Chen, Z., Cao, Y., Liu, Y., Wang, H., Xie, T., & Liu, X. (2020, November). A comprehensive study on challenges in deploying deep learning based software. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 750-762).

Esmaeilzadeh, A. 2017. A Test Driven Approach to Develop Web-Based Machine Learning Applications. UNLV Theses, Dissertations, Professional Papers and Capstones. 3127.

Fitzgerald, B., & Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176-189. https://doi.org/10.1016/j.jss.2015.06.063

Grayson, J. E. (2000). Python and Tkinter programming. Manning Publications Co. Greenwich. http://117.239.19.55:8080/jspui/handle/123456789/230

Geron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: Concepts, tools and techniques to build intelligent systems. O'Reilly Media. ISBN-10:1492032611.

Hull, J. C. (2020). Machine Learning in Business: An Introduction to the World of Data Science. Independently published. https://www.amazon.com/Machine-Learning-Business-Introduction-Science/dp/B088B8162S

IEEE CS. (2000). Recommended Practice for Architectural Description for Software-Intensive Systems. https://doi.org/10.1109/ieeestd.2000.91944

Jaxenter. (2018). ML trends in Stack Overflow Developer Survey 2018. https://jaxenter.com/ml-trends-stack-overflow-145870.html

Keshari, K. (2020, December 02). Top 10 Applications of Machine Learning: Machine Learning Applications in Daily Life. Retrieved from. https://www.edureka.co/blog/machine-learning-applications

Jordon, W. (2019). Python Django Web Development: The Ultimate Django web framework guide for Beginners. Independently published. https://www.amazon.com/Python-Django-Web-Development-framework/dp/1688542817

Kruchten, P., Nord, R. L., Ozkaya, I., & Visser, J. (2012). Technical debt in software development. ACM SIGSOFT Software Engineering Notes, 37(5), 36-38. https://doi.org/10.1145/2347696.2347698

Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101, 193-220. https://doi.org/10.1016/j.jss.2014.12.027

McClendon, L., & Meghanathan, N. (2015). Using Machine Learning Algorithms to Analyze Crime Data. Machine Learning and Applications: An International Journal, 2(1), 1-12. https://doi.org/10.5121/mlaij.2015.2101

McGovern, J., Ambler, S. W., Stevens, M. E., Linn, J., Jo, E. K., & Sharan, V. (2004). A practical guide to enterprise architecture. Prentice Hall Professional. ISBN-10: 0131412752.

Mikut, R., & Reischl, M. (2011). Data mining tools. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 1(5), 431-443. https://doi.org/10.1002/widm.24

Miller, J. (2019). Hands-On Machine Learning with IBM Watson: Leverage IBM Watson to implement machine learning techniques and algorithms using Python. Packt Publishing Ltd. https://www.amazon.com/Hands-Machine-Learning-IBM-Watson/dp/1789611857

Moroney, L. (2020). Ai and machine learning for coders. O'Reilly Media, Incorporated. https://www.oreilly.com/library/view/ai-and-machine/9781492078180/

Murad, D. F. (2020). Systematic Literature Review (SLR) Approach. https://doi.org/10.31219/osf.io/v7239.

O'Leary, K., & Uchida, M. (2020). Common Problems with Creating Machine Learning Pipelines from Existing Code. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/b50bc83882bbd29c50250d1e59fbc3afda3fb5e5.pdf

Oppegaard, S. M. N. (2021). Regulating Flexibility: Uber's Platform as a Technological Work Arrangement. Nordic Journal of Working Life Studies. https://doi.org/10.18291/njwls.122197

Pathak, N. (2017). Artificial Intelligence for. NET: Speech, Language and Search: Building Smart Applications with Microsoft Cognitive Services APIs. Apress. ISBN-10: 1484229495.

Plonski, P. (2019). December 31. Deploy Machine Learning Models with Django. https://www.deploymachinelearning.com/

Ranjeetsingh, S. S. (2014). Microsoft Windows Azure: Developing Applications for Highly Available Storage of Cloud Service. International Journal of Science and Research (IJSR), 4(12), 662-665. https://doi.org/10.21275/v4i12.nov151864

Redapt Marketing. (2019). Why do ML projects fail? https://www.redapt.com/blog/why-90-of-machine-learning-models-never-make-it-to-production#:~:text=During%20a%20panel%20at%20last,actually%20make%20it%20into%20production

Runyu, Xu. (2020). A design pattern for deploying machine learning models to production. https://csusm-dspace.calstate.edu/bitstream/handle/10211.3/217176/XuRunyu_Summer2020.pdf?sequence=1

Sanderson, D. (2012). Programming Google App Engine. Sebastopol, CA: O'Reilly. https://www.oreilly.com/library/view/programming-google-app/9781449314095/

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. Advances in neural information processing systems, 28, 2503-2511. http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems

Sharma, R. (2020). Study of supervised learning and unsupervised learning. International Journal for Research in Applied Science and Engineering Technology, 8(6), 588-593. https://doi.org/10.22214/ijraset.2020.6095

Schröer, C., Kruse, F., & Gómez, J. M. (2021). A Systematic Literature Review on Applying CRISP-DM Process Model. Procedia Computer Science, 181, 526-534. https://doi.org/10.1016/j.procs.2021.01.199

Singh, P. (2021). Deploy machine learning models to production: with flask, streamlit, docker and kubernetes on google cloud platform. Apress. http://103.7.177.7/handle/123456789/207519

Stack Overflow. (2020). We <3 people who code. https://stackoverflow.com/never make it into production? https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/

Wang, Q. (2019). Machine learning applications in operations management and digital marketing (Doctoral dissertation, Universiteit van Amsterdam). https://abs.uva.nl/binaries/content/assets/subsites/amsterdam-business-school/research/dissertations/thesis-q.-wang---abs-2019.pdf

Washizaki, H., Uchida, H., Khomh, F., & Guéhéneuc, Y. G. (2019, December). Studying software engineering patterns for designing machine learning systems. In 2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP) (pp. 49-495). IEEE. https://ieeexplore.ieee.org/abstract/document/8945075/

Zazworka, N., Shaw, M. A., Shull, F., & Seaman, C. (2011, May). Investigating the impact of design debt on software quality. In Proceedings of the 2nd Workshop on Managing Technical Debt (pp. 17-23). https://doi.org/10.1145/1985362.1985366

Zhang, D., & Tsai, J. J. (Eds.). (2005). Machine learning applications in software engineering (Vol. 16). World Scientific. https://doi.org/10.1142/9789812569271_0001

Zhang, T., Gao, C., Ma, L., Lyu, M., & Kim, M. (2019, October). An empirical study of common challenges in developing deep learning applications. In 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE) (pp. 104-115). IEEE. https://doi.org/10.1109/issre.2019.00020