International Journal of Scientific Research in Science, Engineering and Technology
Print ISSN: 2395-1990 | Online ISSN : 2394-4099 (www.ijsrset.com)
doi : https://doi.org/10.32628/IJSRSET229632

Design and Development of Modern day Machine Learning Applications - A Survey

Rohan S Siddeshwara, V Sai Rohit, Arshad Pasha, Aditya S Manakar
Department of Artificial Intelligence and Machine Learning, New Horizon College of Engineering, Bangalore, India

Article Info
Volume 9, Issue 6
Page Number : 251-260

Publication Issue :
November-December-2022

Article History
Accepted : 10 Nov 2022
Published: 28 Nov 2022

ABSTRACT
This paper is an overview of the Machine Learning Operations (MLOps) area. Our aim is to define the operation and the components of such systems by highlighting the current problems and trends. In this context we present the different tools and their usefulness in order to provide the corresponding guidelines. Machine learning operations (MLOps) is quickly becoming a critical component of successful data science project deployment in the enterprise. It's a process that helps organisations and business leaders generate long-term value and reduce risk associated with data science, machine learning, and AI initiatives. Yet it's a relatively new concept; so why has it seemingly skyrocketed into the data science lexicon overnight? This introductory chapter delves into what MLOps is at a high level, its challenges, why it has become essential to a successful data science strategy in the enterprise, and, critically, why it is coming to the forefront now.

Keywords : Machine Learning Operations, Ml Algorithms, MLOps

I. INTRODUCTION

Incorporating machine learning models in production is a challenge that remains from the creation of the first models until now. For years data scientists, machine learning engineers, front end engineers, production engineers tried to find a way to work together and combine their knowledge in order to deploy ready for production models. This task has many difficulties and it is not easy to overcome them. This is why only a small percentage of the ML projects manage to reach production. In the previous years a set of techniques and tools have been proposed and used in order to minimise as much as possible this kind of problems. The development of these tools had multiple targets. Data preprocessing, models' creation, training, evaluation, deployment, and monitoring are some of them. As the field of AI progresses such kind of tools are constantly emerging.

At its core, MLOps is the standardisation and streamlining of machine learning life cycle management. But taking a step back, why does the machine learning life cycle need to be streamlined? On the surface, just looking at the steps to go from business problem to a machine learning model at a very high level, it seems straightforward[1].

For most traditional organisations, the development of multiple machine learning models and their deployment in a production environment are relatively new. Until recently, the number of models

may have been manageable at a small scale, or there was simply less interest in understanding these models and their dependencies at a company-wide level. With decision automation (that is, an increasing prevalence of decision making that happens without human intervention), models become more critical, and, in parallel, managing model risks becomes more important at the top level.

MLOps isn't just important because it helps mitigate the risk of machine learning models in production; it is also an essential component to massively deploying machine learning efforts (and in turn benefiting from the corresponding economies of scale). Going from one or a handful of models in production to tens, hundreds, or thousands that have a positive business impact requires MLOps discipline.

To understand the key features of MLOps, it's essential first to understand how machine learning works and be intimately familiar with its specificities. Though often overlooked in its role as a part of MLOps, ultimately algorithm selection (or how machine learning models are built) can have a direct impact on MLOps processes.

At its core, machine learning is the science of computer algorithms that automatically learn and improve from experience rather than being explicitly programmed. The algorithms analyse sample data, known as training data, to build a software model that can make predictions.

II. Model Development

1. Establishing Business Objectives

The development of a machine learning model typically begins with a business objective, which can be as simple as reducing fraudulent transactions to 0.1% or being able to recognise people's faces in social media photos. Business objectives, by definition, include performance targets, technical infrastructure requirements, and cost constraints; all of these factors can be captured as key performance indicators, or KPIs, allowing the business performance of models in production to be monitored.

It is critical to understand that ML projects are typically part of a larger project that affects technologies, processes, and people. As a result, setting objectives includes change management, which may provide some guidance for how the ML model is to be built. For example, the required level of transparency will strongly influence algorithm selection and may drive the need to provide explanations alongside predictions in order for predictions to be turned into valuable business decisions.

Different Ml Algorithms, Different MLOps Challenges

What all ML algorithms have in common is that they make inferences by modelling patterns in previous data, and the quality and relevance of this experience are critical factors in their effectiveness. They differ in that each type of algorithm has unique characteristics and presents unique challenges in MLOps.

Some ML algorithms are better suited to specific use cases, but governance considerations may also play a role in algorithm selection[2]. Highly regulated environments where decisions must be explained (e.g., financial services) cannot use opaque algorithms such as neural networks; instead, simpler technologies such as decision trees must be used. In many cases, the trade-off is not so much one of performance as it is one of cost. In other words, simpler techniques typically necessitate more expensive manual feature engineering to achieve the same level of performance as more complex techniques.

2. Data Sources and Exploratory Data Analysis

With clear business objectives in place, it is time to bring together subject matter experts and data scientists to begin the journey of developing the ML model[4]. This process begins with the search for appropriate input data. Finding data may appear to be a simple task, but in practice, it can be the most difficult part of the journey.

Because data is required to power ML algorithms, it is always beneficial to gain an understanding of the patterns in data before attempting to train models[7].

Exploratory data analysis (EDA) techniques can aid in the development of hypotheses about the data, the identification of data cleaning requirements, and the process of selecting potentially significant features. EDA can be performed visually for intuitive insight or statistically for greater rigour.

● Data Exploration

When data scientists or analysts consider data sources for training a model, they must first understand the nature of the data. Even the most effective algorithm-trained model is only as good as its training data. A number of issues, such as incompleteness, inaccuracy, inconsistency, and so on, can prevent all or part of the data from being useful at this stage.

Examples of such processes can include:

• Documenting how the data was collected and what assumptions were already made

• Looking at summarising statistics of the data: What is the domain of each column? Are there some rows with missing values? Obvious mistakes? Strange outliers? No outliers at all?

• Taking a closer look at the distribution of the data

• Cleaning, filling, reshaping, filtering, clipping, sampling, etc.

• Checking correlations between the different columns, running statistical tests on some subpopulations, fitting distribution curves

• Comparing that data to other data or models in the literature: Is there some usual information that seems to be missing? Is this data comparably distributed?

Of course, domain knowledge is required to make informed decisions during this exploration. Some anomalies may be difficult to detect without specific knowledge, and assumptions can have unintended consequences for the untrained eye. Industrial sensor data is a good example: unless the data scientist is also a mechanical engineer or equipment expert, they may not understand what constitutes normal versus strange outliers for a specific machine.

3.Feature Engineering and Selection

EDA usually leads to feature engineering and feature selection. The process of transforming raw data from selected datasets into "features" that better represent the underlying problem to be solved is known as feature engineering. "Features" are fixed-size arrays of numbers, as they are the only object that ML algorithms understand. Data cleansing is a component of feature engineering that can account for the majority of the time spent on an ML project.

● Feature Engineering Techniques

The most common is one-hot encoding[6]. Text or image inputs, on the other hand, necessitate more intricate engineering. Deep learning has recently transformed this field by providing models that convert images and text into tables of numbers that ML algorithms can use. These tables are known as embeddings, and they enable data scientists to perform transfer learning because they can be used in domains where they have not previously been trained.

● Feature Selection

When it comes to feature creation and selection, the question of how much and when to stop comes up frequently. Including more features may result in a more accurate model, greater fairness when splitting into more precise groups, or compensation for other useful missing information. However, it has some drawbacks, all of which can have a significant impact on MLOps strategies in the long run

Automated feature selection can help by using heuristics to estimate how important certain features will be for the model's predictive performance. For example, one can examine the correlation with the target variable or quickly train a simple model on a representative subset of the data and then examine which features are the most powerful predictors.

4.Training and Evaluation

After data preparation by way of feature engineering and selection, the next step is training. The process of training and optimising a new ML model is iterative; several algorithms may be tested, features can be automatically generated, feature selections may be adapted, and algorithm hyper parameters tuned. Training is the most computationally intensive step of

the ML model life cycle, in addition to (or in many cases because of) its iterative nature.

When iterating, keeping track of the results of each experiment becomes increasingly difficult. Nothing bothers data scientists more than being unable to recreate the best results because they cannot recall the exact configuration. An experiment tracking tool can greatly simplify the process of remembering the data, selecting features, and model parameters, as well as tracking performance metrics. These allow experiments to be compared side by side, highlighting performance differences.

Choosing the best solution necessitates consideration of both quantitative criteria, such as accuracy or average error, and qualitative criteria, such as the explainability of the algorithm or its ease of deployment[3].

● Experimentation

Experimentation takes place throughout the entire model development process, and usually every important decision or assumption comes with at least some experiment or previous research to justify it.When experimenting, data scientists need to be able to quickly iterate through all the possibilities for each of the model

Fortunately, there are tools to do all of this semi ,automatically, where you only need to define what should be tested (the space of possibilities) depending on prior knowledge (what makes sense) and the constraints (e.g., computation, budget).

Trying all combinations of every possible hyper parameter, feature handling, etc., quickly becomes untraceable. Therefore, it is useful to define a time and/or computation budget for experiments as well as an acceptability threshold for usefulness of the model (more on that notion in the next section).

Fortunately, more and more data science and machine learning platforms allow for automating these workflows not only on the first run, but also to preserve all the processing operations for repeatability. Some also allow for the use of version control and experimental branch spin-off to test theories and then merge, discard, or keep them

● Evaluating and Comparing Models

George E. P. Box, a twentieth century British statistician, once said that all models are wrong, but some are useful. In other words, a model should not aim to be perfect, but it should pass the bar of "good enough to be useful" while keeping an eye on the uncanny valley—typically a model that looks like it's doing a good job but does a bad (or catastrophic) job for a specific subset of cases (say, an underrepresented population).

With this in mind, it's important to evaluate a model in context and have some ability to compare it to what existed before the model—whether a previous model or a rules- based process—to get an idea of what the outcome would be if the current model or decision process were replaced by the new one.

● Choosing Evaluation Metrics

Choosing the proper metric by which to evaluate and compare different models for a given problem can lead to very different models A simple example: accuracy is often used for automated classification problems but is rarely the best fit when the classes are unbalanced (i.e., when one of the outcomes is very unlikely compared to the other). In a binary classification problem where the positive class (i.e., the one that is interesting to predict because its prediction triggers an action) is rare, say 5% of occurrences, a model that constantly predicts the negative class is therefore 95% accurate, while also utterly useless.

To get an idea of how well a model will generalise, that metric should be evaluated on a part of the data that was not used for the model's training (a holdout dataset), a method called cross-testing. There can be multiple steps where some data is held for evaluation and the rest is used for training or optimising, such as metric evaluation or hyper parameter optimisation. There are different strategies as well, not necessarily just a simple split. In k-fold cross-validation, for example, data scientists rotate the parts that they hold out to evaluate and train multiple times. This

multiplies the train ing time but gives an idea of the stability of the metric.

Oftentimes, data scientists want to periodically retrain models with the same algorithms, hyper parameters, features, etc., but on more recent data. Naturally, the next step is to compare the two models and see how the new version fares. But it's also important to make sure all previous assumptions still hold: that the problem hasn't fundamentally shifted, that the modelling choices made previously still fit the data, etc. This is more specifically part of performance and drift monitoring

Unfortunately, there is no one-size-fits-all metric. You need to pick one that matches the problem at hand, which means understanding the limits and trade-offs of the metric (the mathematics side) and their impact on the optimisation of the model (the business side).

5.Version Management and Reproducibility

Discussing the evaluation and comparison of models (for fairness as discussed immediately before, but also a host of other factors) necessarily brings up the idea of version control and the reproducibility of different model versions. With data scientists building, testing, and iterating on several versions of models, they need to be able to keep all the versions straight.

Version management and reproducibility address two different needs:

● During the experimentation phase, data scientists may find themselves going back and forth on different decisions, trying out different combinations, and reverting when they don't produce the desired results. That means having the ability to go back to different "branches" of the experiments—for example, restoring a previous state of a project when the experimentation process led to a dead end.

● Data scientists or others (auditors, managers, etc.) may need to be able to replay the computations that led to model deployment for an audit team several years after the experimentation was first done.

● Reproducibility

While many experiments are transient, significant versions of a model must be saved for future use. The issue at hand is reproducibility, which is a key concept in experimental science in general. The goal of machine learning is to save enough information about the environment in which the model was developed so that it can be recreated with the same results from scratch.

Without reproducibility, data scientists have little chance of confidently iterating on models, and even less chance of handing the model over to DevOps to see if what was created in the lab can be faithfully reproduced in production. True reproducibility necessitates version control of all assets and parameters involved, including training and evaluation data, as well as a record of the software environment.

6.Productionalization and Deployment

Model productionalization and deployment is an important component of MLOps that presents a completely different set of technical challenges than model development[5]. It is the responsibility of the software engineer and the DevOps team, and the organisational challenges in managing information exchange between data scientists and these teams should not be underestimated. Delays or failures to deploy are unavoidable without effective team collaboration.

Model Deployment Types and Contents

To understand what happens during these stages, take a step back and ask yourself: what exactly goes into production, and what does a model consist of? Model deployment is commonly divided into two types:

● Live-scoring model or model-as-a-service

The model is typically deployed in a simple framework to provide a REST API endpoint (the means by which the API can access the resources required to perform the task) that responds to requests in real time.

● Embedded model

Here the model is packaged into an application, which is then published. A common example is an application that provides batch-scoring of requests.

What to-be-deployed models consist of depends on the technology used, but they typically consist of a set of code (typically Python, R, or Java) and data artefacts.

Any of these can have runtime and package version dependencies that must match in the production environment because using different versions may cause model predictions to differ.

One approach to reducing dependencies on the production environment is to export the model to a portable format such as PMML, PFA, ONNX, or POJO. These aim to improve model portability between systems and simplify deployment. However, they come at a cost: each format supports a limited range of algorithms, and sometimes the portable models behave in subtly different ways than the original. Whether or not to use a portable format is a choice to be made based on a thorough understanding of the technological and business context.

● Containerisation

Containerisation is becoming an increasingly popular solution to the problems associated with dependencies when deploying ML models. Container technologies, such as Docker, are lightweight alternatives to virtual machines, allowing applications to be deployed in independent, self-contained environments that are tailored to the exact needs of each model.They also allow for the seamless deployment of new models via the blue-green deployment technique. Model compute resources can also be elastically scaled using multiple containers. The role of technologies such as Kubernetes, which can be used both in the cloud and on-premise, is to orchestrate many containers.

● Runtime Environments

The first step in sending a model to production is making sure it's technically possible ideal MLOps systems favor rapid, automated deployment over labor-intensive processes, and runtime environments can have a big effect on which approach prevails.

Production environments take a wide variety of forms: custom-built services, data science platforms, dedicated services like TensorFlow Serving, low-level infrastructure like Kubernetes clusters, JVMs on embedded systems, etc. To make things even more complex, consider that in some organisations, multiple heterogeneous production environments coexist.

Ideally, models running in the development environment would be validated and sent as is to production; this minimises the amount of adaptation work and improves the chances that the model in production will behave as it did in development. Unfortunately, this ideal scenario is not always possible, and it's not unheard of that teams finish a long-term project only to realise it can't be put in production.

● Adaptation from Development to Production Environments

In terms of adaptation work, on one end of the spectrum, the development and production platforms are from the same vendor or are otherwise interoperable, and the dev model can run without any modification in production. In this case, the technical steps required to push the model into production are reduced to a few clicks or commands, and all efforts can be focused on validation.

On the other end of the spectrum, there are cases where the model needs to be reimplemented from scratch—possibly by another team, and possibly in another programming language. Given the resources and time required, there are few cases today where this approach makes sense. However, it's still the reality in many organisations and is often a consequence of the lack of appropriate tooling and processes. The reality is that handing over a model for another team to reimplement and adapt for the production environment means that model won't reach production for months (maybe years), if at all.

Between these two extreme cases, a number of transformations per formed on the model or interactions with its environment can be made to make it production compatible. In all cases, validation should be performed in an environment that is as close to production as possible, rather than in the development environment.

● Tooling considerations

The format required to send to production should be considered early, as it may have a large impact on the model itself and the quantity of work required to

productionalize it. For example, when a model is developed using scikit-learn (Python) and production is a Java-based environment that expects PMML or ONNX as input, conversion is obviously required.

● Performance considerations

Another common reason conversion may be required is for performance. For example, a Python model will typically have higher latency for scoring a single record than its equivalent converted to C++. The resulting model will likely be dozens of times faster (although obviously it depends on many factors, and the result can also be a model that is dozens of times slower).

● Data Access Before Validation and Launch to Production

In some cases, data can be frozen and bundled with the model. But when this is not possible (e.g., if the dataset is too large or the enrichment data needs to always be up to date), the production environment should access a database and thus have the appropriate network connectivity, libraries, or drivers required to communicate with the data storage installed, and authentication credentials stored in some form of production configuration.

Managing this setup and configuration can be quite complex in practice since, again, it requires appropriate tooling and collaboration (in particular to scale to more than a few dozen models). When using external data access, model validation in situations that closely match production is even more critical as technical connectivity is a common source of production malfunction.

7.Model Deployment Requirements

So, what needs to be addressed in the productionalization process between completing model development and physically deploying into production? One thing is certain: rapid, automated deployment is always preferable to time-consuming processes.

Testing and validation are frequently unnecessary for short-lived, self-service applications. If the model's maximum resource demands can be safely capped by technologies like Linux groups, then a fully automated single-step push-to-production may be perfectly adequate. When using this lightweight deployment mode, it is even possible to handle multiple user interfaces with frameworks such as Flask. In addition to integrated data science and machine learning platforms, some business rule management systems may allow for the automatic deployment of basic ML models.

In customer-facing, mission-critical use cases, a more robust CI/CD pipeline is required. This typically involves:

1. Ensuring all coding, documentation and sign-off standards have been met
2. Re-creating the model in something approaching the production environment
3. Revalidating the model accuracy
4. Performing explainability checks
5. Ensuring all governance requirements have been met
6. Checking the quality of any data artefacts
7. Testing resource usage under load
8. Embedding into a more complex application, including integration tests

8.Monitoring

Once a model is deployed to production, it is crucial that it continue to perform well over time. But good performance means different things to different people, in particular to the DevOps team, to data scientists, and to the business.

● DevOps Concerns

The DevOps team's concerns are well-known, and they include questions such as:

1. Is the model doing the job quickly enough?
2. Is it utilising an appropriate amount of memory and processing time?

This is traditional IT performance monitoring, and DevOps teams are already adept at it. In this regard, the resource requirements of ML models are not dissimilar to those of traditional software.

The scalability of computing resources is important to consider, Overall, the existing expertise in DevOps teams for resource monitoring and management can be easily applied to ML models.

● Data Scientist Concerns

The real world does not stop. The training data used to build a fraud detection model six months ago will not reflect a new type of fraud that began in the last three months. If a website begins to attract a younger user base, a model that generates advertisements is likely to produce less and less relevant advertisements. At some point, the performance will become unacceptable, necessitating model retraining. How frequently models must be retrained is determined by how quickly the real world changes and how accurate the model must be, but also, and most importantly, by how simple it is to build and deploy a better model.

● Ground truth

The ground truth, put simply, is the correct answer to the question that the model was asked to solve.,In knowing the ground truth for all the predictions a model has made, one can judge how well that model is performing. Sometimes ground truth is obtained rapidly after a prediction

● Input drift

Input drift is based on the idea that a model can only predict accurately if the data it was trained on is an accurate representation of the real world. So, if a comparison of recent requests to a deployed model against training data reveals significant differences, then the model performance is likely to be compromised. This is the foundation for monitoring input drift. The beauty of this approach is that all of the data needed for this test already exists, eliminating the need to wait for ground truth or other information. Identifying drift is one of the most important components of an adaptable MLOps strategy, and one that can bring agility to the organisation's enterprise AI efforts overall.

● Iteration and Life Cycle

Developing and deploying improved versions of a model is an important and difficult part of the MLOps life cycle. One of the reasons for developing a new model version is model performance degradation due to model drift, as discussed in the preceding section. Sometimes there is a need to reflect refined business objectives and KPIs, and other times the data scientists have simply developed a better way to design the model.

Iteration

In some industries with a high rate of change, new training data is made available daily. Daily retraining and redeployment of the model are frequently automated to ensure that the model accurately reflects recent experience.

The most basic scenario for iterating a new model version is retraining an existing model with the most recent training data. However, even though there are no changes to the feature selection or algorithm, there are still numerous pitfalls. More specifically:

● Does the new training data look as expected? Automated validation of the new data through predefined metrics and checks is essential.

● Is the data complete and consistent?

● Are the distributions of features broadly similar to those in the previous training set? Remember

● that the goal is to refine the model, not radically change it.

After creating a new model version, compare the metrics to the current live model version. To do so, both models must be evaluated on the same development dataset, whether the previous or latest version. If metrics and checks show that the models differ significantly, automated scripts should not be reused, and manual intervention should be sought.

● Even in the "simple" automated retraining scenario with new training data, there is a need for multiple development datasets based on scoring data reconciliation (with ground truth when it becomes available), data cleaning and validation, the previous model version, and a set of carefully considered checks. Retraining in other scenarios is likely to be even more

complicated, rendering automated redeployment unlikely.

The Feedback Loop

DevOps best practices usually dictate that the live model scoring environment and the model retraining environment be separate in large enterprises. As a result, the evaluation of a new model version on the retraining environment is very likely.

One approach to mitigating this uncertainty is shadow testing, where the new model version is deployed into the live environment alongside the existing model. All live scoring is handled by the incumbent model version, but each new request is then scored again by the new model version and the results logged, but not returned to the requestor. Once sufficient requests have been scored by both versions, the results can be compared statistically. Shadow scoring also gives more visibility to the SMEs on the future versions of the model and may thus allow for a smoother transition.

III. Summary

Putting machine learning models into production is a significant challenge for many organisations. As AI initiatives expand, MLOps is the cornerstone in ensuring deployed models are well maintained, perform as expected, and do not adversely affect the business. Therefore, proper MLOps practices are essential.

While we can run standard software in production for years without updating it, this is far from realistic for a machine learning (ML) model. There is an inherent decay in model predictions that requires regular retraining. Managing these updates manually quickly becomes tedious and is not scalable. Automation begins with identifying which metrics to monitor, when these metrics become worrisome, and what indicators are used to determine whether a new version of a model is outperforming the current version.

These challenges highlight the importance of seeing MLOps as a complete puzzle with the pieces coming from designing, building, deploying, monitoring, and governing models.

In this paper we have discussed how we can handle some of these issues in developing modern day ML solutions.

MLOps Principles	**Data**	**ML Model**	**Code**
Documentation	1) Data sources 2) Decisions, how/where to get data 3) Labelling methods	1) Model selection criteria 2) Design of experiments 3) Model pseudo-code	1) Deployment process 2) How to run locally
Project Structure	1) Data folder for raw and processed data 2) A folder for data engineering pipeline 3) Test folder for data engineering methods	1) A folder that contains the trained model 2) A folder for notebooks 3) A folder for feature engineering 4)A folder for ML model engineering	1) A folder for bash/shell scripts 2) A folder for tests 3) A folder for deployment files (e.g Docker files)
Versioning	1) Data preparation pipelines 2) Features store 3) Datasets 4) Metadata	1) ML model training pipeline 2) ML model (object) 3) Hyperparameters 4) Experiment tracking	1) Application code 2) Configurations
Testing	1) Data Validation (error detection) 2) Feature creation unit testing	1) Model specification is unit tested 2) ML model training pipeline is integration tested 3) ML model is validated before being operationalized 4) ML model staleness test (in production) 5) Testing ML model relevance and correctness 6) Testing non-functional requirements (security, fairness, interpretability)	1) Unit testing 2) Integration testing for the end-to-end pipeline

Automation	1) Data transformation 2) Feature creation and manipulation	1) Data engineering pipeline 2) ML model training pipeline 3) Hyperparameter /Parameter selection	1) ML model deployment with CI/CD 2) Application build
Reproducibility	1) Backup data 2) Data versioning 3) Extract metadata 4) Versioning of feature engineering	1) Hyperparameter tuning is identical between dev and prod 2) The order of features is the same 3) Ensemble learning: the combination of ML models is same 4)The model pseudo-code is documented	1) Versions of all dependencies in dev and prod are identical 2) Same technical stack for dev and production environments 3) Reproducing results by providing container images or virtual machines
Deployment	1) Feature store is used in dev and prod environments	1) Containerization of the ML stack 2) REST API 3) On-premise, cloud, or edge	1) On-premise, cloud, or edge
Monitoring	1) Data distribution changes (training vs. serving data) 2) Training vs serving features	1) ML model decay 2) Numerical stability 3) Computational performance of the ML model	1) Predictive quality of the application on serving data

IV.REFERENCES

[1] S. Makinen, H. Skogstr ¨ om, V. Turku, E. Laaksonen, and ¨ T. Mikkonen, "Who needs mlops: What data scientists seek to accomplish and how can mlops help?"

[2] C. Renggli, L. Rimanic, N. M. Gurel, B. Karla ¨ s, W. Wu, ˇ C. Zhang, and E. Zurich, "A data quality-driven view of mlops," 2 2021. [Online]. Available: https://arxiv.org/abs/ 2102.07750v1

[3] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam, "Demystifying mlops and presenting a recipe for the selection of open-source tools," Applied Sciences 2021, Vol. 11, Page 8861, vol. 11, p. 8861, 9 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/ 19/8861/htmhttps://www.mdpi.com/2076-3417/11/19/8861

[4] J. Klaise, A. V. Looveren, C. Cox, G. Vacanti, and A. Coca, "Monitoring and explainability of models in production," 7 2020. [Online]. Available: https://arxiv.org/abs/2007.06299v1

[5] S. Alla and S. K. Adari, "What is mlops?" in Beginning MLOps with MLFlow. Springer, 2021, pp. 79–124.

[6] Introducing MLOps "How to Scale Machine Learning in the Enterprise " by Mark Treveil and the Dataiku Team

[7] E. RAJ, "Mlops using azure machine learning rapidly test, build, and manage production-ready machine learning life cycles at scale." PACKT PUBLISHING LIMITED, pp. 45–62, 202

Cite this article as :

Rohan S Siddeshwara, V Sai Rohit, Arshad Pasha, Aditya S Manakar, "Design and Development of Modern day Machine Learning Applications - A Survey ", International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Online ISSN : 2394-4099, Print ISSN : 2395-1990, Volume 9 Issue 6, pp. 251-260, November-December 2022. Available at doi : https://doi.org/10.32628/IJSRSET229632

Journal URL : https://ijsrset.com/IJSRSET229632