2023 International Conference on Data Science and Network Security (ICDSNS)

Cost Effective Generic Machine Learning Operation: A Case Study

1st Samridhi Jain
Department of Computer science & Engineering,
Chandigarh University
Mohali, India
Samridhijain100@gmail.com

2nd Puneet Kumar
Department of Computer science & Engineering
Chandigarh University
Mohali, India
Professor.pkumar@gmail.com

***Abstract*—In this research, we have proposed a mechanism to implement a typical Mops pipeline for small scale organization who cannot afford the operational expenditures to bring the pipeline at Cloudera, Horton works platform or cloud premises like AWS, GCP or Azure. This paper gives a very detailed understanding of operationalization of a typical ML pipelines to adhere all the elements and artifacts without even using any Docker, Kubernetes or even any API generating platforms like Flask or FastAPI. Using the combination of a simple Python/R along with SQL and Shell scripts we can manage the entire workflow at on premises with a very low-cost approach. From some angle this mechanism would not be comparable with the architectures like market ready MLOps platforms like Azure Devops, MLflow, Kubeflow, Apache Airflow, Databricks with Data factory or Sagemaker Studio workflow but from conceptual point of view, suffice almost 90% of the requirements with efficient manner. We have also done a latest review related to MLOps in recent past and also listed out the several research gaps that can be solved in future research.**

Keywords—MLOps, ML pipeline, cloud platform, Low Budget Architecture, ML Engineering

I. INTRODUCTION

In current Data science and Analytics business landscape MLOps is a very prominent and fascinating concept that most of the organizations are very optimistic to implement as a part of their AIML initiatives [1]. Different level of organizations across different lines of business has their own policy to adopt the MLOps to orchestrate the workflow. Large scale organizations can easily afford expensive licenses or operational expenses to bring their entire ML pipeline in customized platforms like Cloudera Manager CDP, CML or SDX or using the Cloud PaaS or IaaS [2]. There are gamut's of state-of-the-art options to consume the developed ML pipelines and push the same in real production environment and post deployment monitoring and tracking without many difficulties of creating self-made architectures and arrangements [12].

But if we see the other angles of the industry, there are lot of small players in the market or start-up organization who are nurturing their end-to-end journey with custom architectures and arrangements and without affording the expensive SOTA platforms for operationalizing their ML pipelines. [3]

Consider a typical MLOps journey and why its required to implement for a successful business solution [4]. To answer the points lets understand the context of the following questions?

- Where do we capture the data and does the single version of data always suffices to meet the requirement?
- How do we manage the different versions of codes that all analyst/Data scientist creates for different iterations?
- How do we capture the Model artifacts for so many numbers of iterations and a reference point for future comparison?
- How do we capture the model logs to track the performance and deployment status?
- What we do if the model performance degrades after few iterations post deployment?
- How do we monitor the performance of the model in case there are a few lines up for the same utilization?
- How do we profile the Model infrastructure for a seamless production execution based upon the business requirement?
- How do we capture the test iterations results?
- How many times we can manually changes the pipelines and execute them once there are minor/major changes in the code?
- Who will take the initiative to maintain the release of the pipelines in case there are some alterations, and the latest version should be in place?

All these points are very much relevant in today's AIML landscape to successfully implement and maintain a solution in production environment. In the next section II, related works is described in today's Section III is devoted to MLOps workflow with some hands-on custom architectures, its benefits, limitations. Finally in section IV, two sample use cases are presented that explain the structure of deployment with scalable support towards the requirements. This section is followed by conclusion and discussions.

II. RELATED WORKS

The article[9] discusses the importance of managing expectations in machine learning (ML) projects and provides tools and techniques for setting realistic project milestones. The paper emphasizes the need for clear communication and understanding between stakeholders to ensure successful ML project outcomes.

Another research [10] explores MLOps, focusing on continuous delivery and automation pipelines in machine learning. The work highlights the significance of automating ML workflows, versioning models and data, and deploying models at scale. It discusses best practices and challenges in implementing MLOps methodologies.

Sculley et al. (2015) shed light on hidden technical debt in machine learning systems, emphasizing the importance of

2023 International Conference on Data Science and Network Security (ICDSNS) | 979-8-3503-0159-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDSNS58469.2023.10245408

code maintainability, testability, and scalability. The paper raises awareness about the challenges that arise due to the rapid evolution of ML models and the need for ongoing code maintenance and monitoring. [11]

Gift and Deza (2021) present practical insights and lessons learned from implementing machine learning systems, covering various aspects such as data preprocessing, model training, deployment, and monitoring. The work provides recommendations and guidelines to address challenges faced during the ML lifecycle.

In a related article [13] introduce the data science lifecycle, providing a comprehensive overview of the stages involved in a data science project, including data collection, preprocessing, modeling, evaluation, and deployment. The paper discusses the importance of collaboration, reproducibility, and documentation in ensuring the success of data science endeavors.

Vartak et al. (2016) present ModelDB, a system for managing machine learning models. The work focuses on the challenges associated with model management, including model versioning, performance tracking, and collaborative model development. ModelDB provides a solution to track and organize ML models for reproducibility and efficient collaboration.[14]

Soh and Singh (2020) discuss the rise of machine learning operations as a field dedicated to managing the production deployment and maintenance of ML models. The paper explores the key concepts, challenges, and benefits of MLOps, highlighting its role in improving model reliability, scalability, and interpretability.[15]

Polyzotis, Roy, Whang, and Zinkevich (2017) present a survey on challenges and opportunities in machine learning model management. The work covers various aspects such as model versioning, deployment, monitoring, and governance. It provides insights into the complexities of managing ML models at scale and offers suggestions to address those challenges.[16]

Sato et al. (2019) delve into the concept of continuous delivery for machine learning, drawing parallels between software engineering and ML workflows. The work explores techniques and best practices for automating the ML pipeline, enabling faster iterations, and facilitating the integration of ML models into production systems.[17]

Zaharia et al. (2018) introduce MLflow, a platform for managing the ML lifecycle. The work presents an open-source tool that enables tracking experiments, packaging code, and managing model deployments. MLflow aims to streamline the ML workflow and enhance collaboration among data scientists and engineers.[18]

While MLOps (Machine Learning Operations) has gained significant attention and research interest, there are still several research gaps that exist. Here are some research gaps in MLOps:

1) Interpretability and Explainability: Enhancing the interpretability and explainability of machine learning models deployed in production systems is a key research area. Developing techniques to provide transparent and understandable insights into model behavior and decision-making processes is crucial for building trust and addressing regulatory requirements. [19]

2) *Data Versioning and Lineage:* While model versioning is well-studied, there is a need for more research on data versioning and lineage in MLOps. Developing effective mechanisms to track and manage changes to data over time, ensuring reproducibility, and enabling proper auditing and governance of data pipelines is essential. [20]

3) Human-in-the-Loop Integration: Research is needed to better understand how to effectively integrate human expertise and feedback into the MLOps process. Developing frameworks and methodologies to leverage human input in model development, validation, and monitoring can enhance model performance and address ethical concerns. [21]

4) Dynamic Model Updating and Continuous Learning: Enabling dynamic model updates and continuous learning in production systems is a research area that requires further exploration. Developing techniques to update models seamlessly without interrupting system operations, and effectively incorporating new data to improve model performance, is critical for real-time adaptation and optimization.[22]

5) Ethical Considerations and Bias Mitigation: There is a need for research on ethical considerations and bias mitigation in MLOps. Developing techniques to identify and mitigate bias in data, models, and decision-making processes, as well as ensuring fairness and transparency, is important for responsible and unbiased AI systems. [23]

6) *Hybrid and Distributed MLOps Architectures:* Research is needed to explore hybrid and distributed MLOps architectures. This includes investigating the challenges and benefits of deploying models in edge computing environments, federated learning approaches, and hybrid cloud-edge systems. [24]

7) Automated Hyperparameter Optimization and AutoML: Research can focus on automating hyperparameter optimization and model selection processes in MLOps. Developing efficient algorithms and methodologies to automate the selection and configuration of models, as well as hyperparameter tuning, can streamline the model development and deployment pipeline.[25]

8) *Cost Optimization and Resource Allocation:* Optimizing the cost and resource allocation in MLOps is an important research area. Developing techniques to automatically scale resources based on demand, optimize compute utilization, and allocate resources efficiently can lead to cost-effective and scalable MLOps solutions. [26]

9) End-to-End MLOps Security and Privacy: Research is needed to address the security and privacy challenges in end-to-end MLOps workflows. This includes exploring techniques to protect sensitive data, secure model deployments, prevent adversarial attacks, and ensure compliance with privacy regulations. [27]

10) MLOps for Emerging Domains: Further research is needed to explore MLOps applications in emerging domains such as healthcare, autonomous systems, robotics, and IoT. Understanding the unique challenges, requirements, and potential benefits of MLOps in these domains can drive advancements and innovations. [28]

Addressing these research gaps will contribute to advancing the field of MLOps, enabling more robust, interpretable, and trustworthy machine learning systems in production environments.

III. Elements Of A Typical MLOPS Pipeline[13]

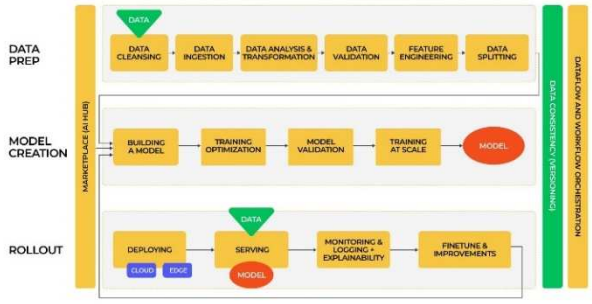

Fig. 1. A typical MLOps flow

A typical MLOps flow comprises the above-mentioned steps in Fig. 1 and here are the details.

1) *Code Management:* Code versioning and maintenance
2) *Data Management:* Versioning, Incremental or Full Load, Central Location
3) *Model Building:* Log All metrices, Logs and artifacts, organizing Multiple iterations, Central locations, Meta data and Data Lineage, Provisioning Infrastructure
4) *Model Management:* Central Location, Versioning, Logs, Outputs, Hyper Parameters, plots, Meta Data
5) *Model Profiling:* Memory and processor requirements.
6) *Model Deployment:* Automated deployment, provisioning infrastructure, End point versioning, Traffic Management (scale up and scale out)
7) *Smoke Test:* End Point Health, Positive and Negative Unit tests
8) *Data Capture:* Real time input and prediction capture, Central location and time capture
9) *Governance:* Model approve/Rejection, Notifications
10) *Model Drift:* fluctuation on expected Model outcome
11) *Data Drift:* Unexpected feature behavior change
12) *Model Update:* Keep the model update, retraining, versioning based on latest dataset and code integration.

The entire flow is also embedded with three promising concepts like Continuous Integration (CI), Continuous Deployment (CD) and Continuous training (CT) pipelines [5]. These is the standard structure of a desired MLOps structure and now lest explain how these requirements can be served using a custom code-based architecture and how much we can achieve without using any state-of-the-art architectures.

A. Core Concept

Few of the core components of the architecture and its application:

1) *Centralized Feature Repository:* There is a single data repository which also comprises multiple Feature marts that also comprises different work profiles and, in a nutshell, these are the unique source of incremental load where the multiple feature marts combined to generate the live data for the model consumption. There would be a feature of data drift calculation that we will discuss in later phase. For future reference also the repository maintains the data versioning which used for each iteration.
2) *Model repository:* This is a centralized repository of all the model files that are generated by the data scientists and based on the version number the required one can be accessed from the repository. This is also a serve-based repository and the number of model, and its size also depends upon the server space and capacity.
3) *Source Code repository:* All the required codes like data preparation and transformation SQL codes, Model execution or scoring codes on Python or R, dependency codes and config files, Shell scripts that also merge few of the transformation and model execution codes in a unique flow. In a nutshell this is a centralized location to cater all the source codes individually or through a pipeline.
4) *Computation Platform:* This is the engine where all the codes will execute, it could be a UNIX based platform or an Anaconda environment or a RStudio server. The processing capacity would be defined and varied based upon the serve memory and space allotted for the specific job.
5) *Continuous Integration:* There would be separate config files like. YAML which caters all the parameters details and, in the release, codes will be parametric and call the config files during the execution. Data scientist will only make the changes in the config files and in subsequent execution the changes will reflect in the base code. Hence in a very manual way the integration can be taken place.
6) *Continuous Deployment*: There would be the shell scripts which keep executing the pipelines through a CRON scheduler on periodic basis. Whenever the YAML files upload the changes automatically updates in the base code and during the pipeline execution the change will hit the model files. Hence, we are not configuring any trigger based continuous deployment, but this is a periodic deployment to keep the model files updated with respect to any changes in the base code.
7) *Continuous Training:* This concept will also cover separately in a different section but already taken care in this architecture.
8) *Model Drift:* This is an embedded part of the Continuous training pipeline and also be explained in a subsequent section.
9) *Data Drift*: This is also an individual topic that we will discuss in a separate section.
10) *Governance*: The logging mechanism will be there to monitor and notify against all the key performance and selection/rejection of the specific version of the models and the final acceptance criteria.
11) *Visualization/Result or Drift showcasing:* For this purpose, we are not using any third-party tools like Tableau or power BI. In python own capability, we can use Dtale or Sweetviz to create interactive visualization within Jupyter environment and can drag and drop features to showcase a presentable outcome.

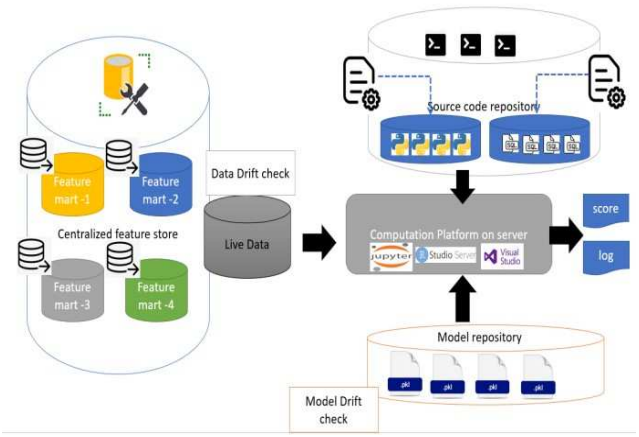

Fig. 2. A typical Model operation workflow [18]

The flow in the above Fig. 2 is depicting that the core computational platform will call the latest model files based on the user's discretion and the specific code version from the code repository which is already integrated. Finally, it consumes the live data block from the unified feature store and then complete the execution in the platform and generates the output file through a csv, a log file and the model artifacts and relevant metrices with .txt file which will store in a model output repository for future references.

The model execution workflow covers the versioning and end to end integration and entire pipeline would be embedded through a shell script and the same will scheduled for periodic execution based upon the business requirement.

B. *Model Drift & Continous Training Pipeline*

This is a very crucial step, and a step-by-step elaboration as shown in Fig. 3 is given below:

1. From the data repository (unique feature store the live data block would be ready for the current timestamp – Window T)
2. Using the SQL transformation, the processed data is ready for the model consumption, that is also for the same time window T.
3. Now the shell script will make the execution and compare the results with the benchmark or with the result of last execution (T-1) and (T-2). In case there is any significant changes we call this as model drift.
4. This triggers another pipeline were consuming the processed data at Time window T a retrainable python script will execute and generate a new model version (Time window T).
5. Now compare the KPIs of the model of Time window T along with the model of Time window (T1) and (T-2) and then decide which one would be productionized at that moment. This type of model deployment is called a BLUE GREEN deployment and finally based upon the comparison (A/B testing) it's been decided whether we will go for the blue model or Green one.
6. Then the final model would deploy on the server and the subsequent scoring will happen from these models itself.

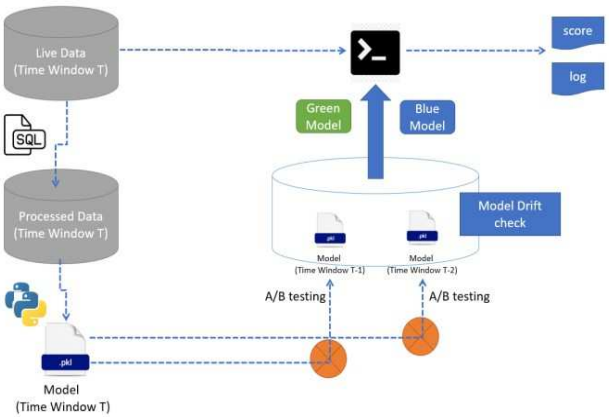

Fig. 3. CT pipeline with model drift calculation

This is how the model drift monitoring will happen along with the continuous training. This is a continuous iterative process and whenever there would be a significant drift and new live data block will hit the pipeline the same exercise will happen and update the model in the target sever.

C. *Data Drift & Impact Analysis*

This is a very interesting solution to identify and visualize the data drift and subsequent impact analysis. Here the final training will generate the model file and the list of important parameters along with degree of importance. One example of this process is shown in Fig. 4. Here we can find through analysis the % change in the data (as highlighted in the red color). If change is exceeding a certain limit then we can find the data drift and its impact to control the change.

Once there would be significant drift in the data for those important parameters then only this will impact the model performance. In case there are huge drift with some parameters which is not at all a driver parameter for the model will not cause much fluctuation in the model performance.

There are generally two different types of parameter – numeric and categorical. For categorical we will simply check the distribution of the lagged version and the current version and interpret the same visually using the Pandas profiling, SweetViz or Dtale platform.

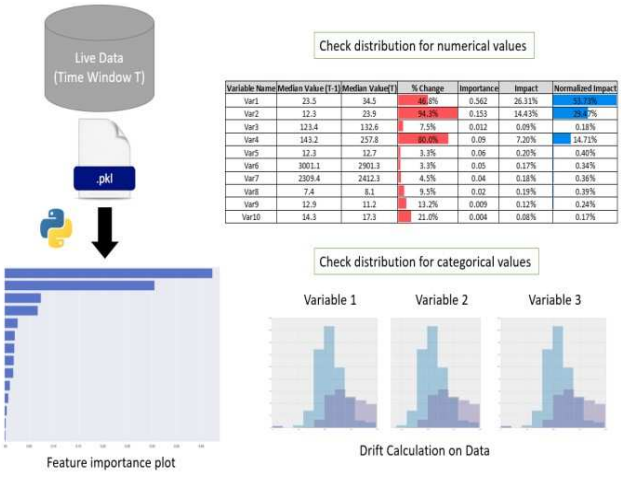

Fig. 4. Data drift engagement

For the numeric variables' median is a better statistical benchmark and we will compare the same for the latest data with respect to the last version data and plot the drift based on

the distribution median of the numeric values. Now we will multiple the model importance ratio (normalized one) along with the drift parameters and calculate the impact.

Thereby its very robust solution to estimate the impact caused by each parameters drift and the same can be logged for all iterations for future references.

D. *Sample Use Case for the MLOps Architecture*

Mostly all predictive Models, Regression problems, Time series forecasting, Optimization problems using batch data can be fitted in this mechanism properly where elasticity/scalability/load balancing is not a concern.

There are few other types of use case that demands real time results or distributed processing for massive performance issues or high-end cognitive vision or NLP use cases or use cases that requires massive scalability/elasticity concern – cannot be suitable for this type of vanilla MLOps architecture.

In those cases, we have no other choice other than using the so-called state of the art MLOps platforms mostly from all three major cloud providers like GCP – Kubeflow, Azure Devops or Databricks MLflow or AWS Sagemaker studio. The heavy computation pytorch, ONNX or TensorFlow type of computation doesn't fit properly with this type of architecture which actually built to cater the generic machine learning problems for small scale industry.

E. *Benefits [18]*

- Absolutely low computation cost in terms of memory, tools, licenses and application platforms.
- Maintenance is easy because the underlying structure is easy, anyone can easily adopt the same.
- No need for any GPU/TPU type execution.
- Very much transparent and explainable solution, no hidden state for the entire pipeline.
- Very much use full for the novice data scientist because the simple Python, R, SQL and shell script will suffice 95% of the requirements.
- No cost for any third-party visualization.
- Can cater big chunk data using Modin pandas, Dask or Vaex (Python inbuilt capabilities)
- Use Python inbuilt parallelism using all cores of the server to distribute the processing capability without involving any spark or cloud capability.
- Easily integrated with any on-premises databases like SSMS, Oracle SQL, SQL server, Teradata.

F. *Limitations [18]*

There are few limitations as well to use the architecture:

- Not useful for any TensorFlow, Pytorch or complex NLP task
- Not suitable for highly scalable solutions
- Cannot used for Real time requirements.
- Also, limitation where TBs of data load is involved.
- Cannot useful once data source be in HDFS, snowflake, RDS, Redshift, Athena, S3, EMR, BigQuery, BigTable, cosmos DB, Synapse or ADLS gen2.

IV. A Case Study On MLOps

To prove our point on MLOps, we have used an example in the form of a case study of animal classification problem where six different types of animals are categorized and classified by using deep learning model. 13412 images belonging to 6 classes for training. 2549, 1845 images are taken for model validation and testing. Fig. 4 indicates the the code snippet from ML pipeline. VGG19 model is used for training on animal dataset. Training accuracy is 0.8975 while test accuracy is 0.7387 in 20 epochs as shown in the Fig. 5.

```
pipe = Pipeline(
    steps=[("scaler", StandardScaler()),
        ("model", Vgg19())
      )
pipe.fit(train_features,train_labels)
```

Fig. 5. A code snippet from ML pipeline

The following pipeline was used for MLOps:

The code snippet in Fig. 5 demonstrates the usage of scikit-learn's Pipeline class for building a machine learning pipeline with two steps: data scaling using StandardScaler and model training using the Vgg19 model.

The Pipeline is initialized with a list of steps, where each step is represented as a tuple containing a name and an instance of an estimator or transformer. In this case, the first step is named "scaler" and is associated with the StandardScaler() transformer, which is used for feature scaling. The second step is named "model" and is associated with the Vgg19() model, which is presumably a deep learning model for image classification. The fit() method is called on the pipeline object, passing the training features (train_features) and corresponding labels (train_labels).

The pipeline orchestrates the execution of the steps in order, applying the transformations (scaling) from the first step and then fitting the model (Vgg19) on the transformed data. By using the Pipeline class, you can combine multiple steps into a single entity, making it easier to organize and execute your machine learning workflow [6-8]. The pipeline ensures that the data transformations and model fitting are performed in a consistent and sequential manner.

Although this is a trivial pipeline for the case study, but it shows the ease of implementing and deployment of ML projects in the applications.

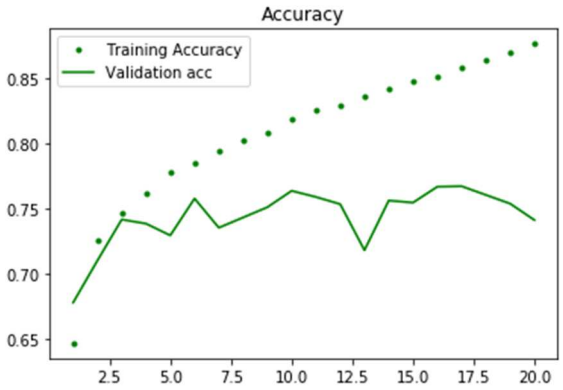

Fig. 6. Training Accuracy for VGG19

V. CONCLUSION

Finally, the solutions have a solid background for lot of small-scale industries or even large scales those uses before 10 years where there was no existence of these state of art MLOps platforms. In some of the cases its still very much effective and cost savings but in some cases, it has some limitations. This is a very custom solution and be adjusted based upon the requirements whereas we cannot adjust the architecture or workflow of any SOTA platforms for our own ease. Definitely lot of additional coding involves implementing the structure for an end-to-end solution, The main objective of sharing the architecture for those novice data scientists to test the framework and gain some in-depth knowledge of how MLOps is actually working rather than using a drag and Drop DAG platforms which is a grey box or black box mechanism doesn't actually explains the core processing at very ground level. I would really encourage junior practitioners to test the architectures on your own hands to get a better flavor of the core MLOps blocks. Through a case study on MLOps we have shown how to use a MLOps on a classification problem.

REFERENCES

[1] Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access.

[2] Recupito, G., Pecorelli, F., Catolino, G., Moreschini, S., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2022, August). A Multivocal Literature Review of MLOps Tools and Features. In 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 84-91). IEEE.

[3] Bodor, A., Hnida, M., & Najima, D. (2023, March). MLOps: Overview of Current State and Future Directions. In Innovations in Smart Cities Applications Volume 6: The Proceedings of the 7th International Conference on Smart City Applications (pp. 156-165). Cham: Springer International Publishing.

[4] Matsui, B. M., & Goya, D. H. (2022, May). MLOps: A Guide to its Adoption in the Context of Responsible AI. In 2022 IEEE/ACM 1st International Workshop on Software Engineering for Responsible Artificial Intelligence (SE4RAI) (pp. 45-49). IEEE.

[5] Kumara, I., Pecorelli, F., Catolino, G., Kazman, R., Tamburri, D. A., & Van Den Heuvel, W. J. (2023, March). Architecting MLOps in the Cloud: From Theory to Practice. In 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C) (pp. 333-335). IEEE.

[6] Calefato, F., Lanubile, F., & Quaranta, L. (2022, September). A preliminary investigation of MLOps practices in GitHub. In Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 283-288).

[7] Filippou, K., Aifantis, G., Papakostas, G. A., & Tsekouras, G. E. (2023). Structure Learning and Hyperparameter Optimization Using an Automated Machine Learning (AutoML) Pipeline. Information, 14(4), 232.

[8] Scriven, A., Kedziora, D. J., Musial, K., & Gabrys, B. (2022). The Technological Emergence of AutoML: A Survey of Performant Software and Applications in the Context of Industry. arXiv preprint arXiv:2211.04148.

[9] https://neptune.ai/blog/life-cycle-of-a-machine-learning-project accessed on march 23, 2023

[10] Symeonidis, G., Nerantzis, E., Kazakis, A., & Papakostas, G. A. (2022, January). MLOps-definitions, tools and challenges. In 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0453-0460). IEEE..

[11] 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..

[12] Gift, N., & Deza, A. (2021). Practical MLOps. " O'Reilly Media, Inc.".

[13] Boenig-Liptsin, M., Tanweer, A., & Edmundson, A. (2022). Data Science Ethos Lifecycle: Interplay of ethical thinking and data science practice. Journal of Statistics and Data Science Education, 30(3), 228-240.

[14] Vartak, M., Subramanyam, H., Lee, W. E., Viswanathan, S., Husnoo, S., Madden, S., & Zaharia, M. (2016, June). ModelDB: a system for machine learning model management. In Proceedings of the Workshop on Human-In-the-Loop Data Analytics (pp. 1-3).

[15] Soh, J., Singh, P., (2020). Machine learning operations. Data Science Solutions on Azure: Tools and Techniques Using Databricks and MLOps, 259-279..

[16] Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2017, May). Data management challenges in production machine learning. In Proceedings of the 2017 ACM International Conference on Management of Data (pp. 1723-1726)..

[17] Sato, D., Wider, A., & Windheuser, C. (2019). Continuous delivery for machine learning. Martin Fowler, 9.

[18] Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... & Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull., 41(4), 39-45.

[19] Zhang, D., Song, Y., Li, Y., Zhang, X., Song, Y., & Song, J. (2020). Towards Explainable and Trustworthy AI for Healthcare: A Survey. *Journal of Healthcare Engineering*, 2020, 1-15.

[20] Veit, A., Albers, A., Mellor, J., & Görnitz, N. (2020). Data Versioning for Machine Learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 3236-3237.

[21] Whitehill, J., Wu, T., Bergsma, J., Movellan, J. R., & Ruvolo, P. (2020). Human-in-the-Loop Machine Learning: Challenges and Opportunities. *ACM Transactions on Intelligent Systems and Technology*, 12(5), Article 58.

[22] Xu, B., Wei, W., Gao, Y., & Chen, E. (2021). Dynamic Model Updating in Production: A Survey. *IEEE Transactions on Big Data*, 7(3), 1075-1094.

[23] Zhao, F., Suhara, Y., Wang, K., & Li, H. (2020). Machine Learning Fairness: A Survey. *ACM Transactions on Management Information Systems*, 11(2), Article 12.

[24] Li, Y., Zhang, S., Guo, D., Chen, J., Zhang, H., Liu, T. Y., & Zhu, Z. (2020). Edge Intelligence: Paving the Last Mile of Artificial Intelligence. *Proceedings of the IEEE*, 108(9), 1454-1485.

[25] Jin, Q., Song, H., & Hu, X. (2020). AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*, 194, Article 105591.

[26] Zhang, X., Wang, S., Sun, J., Wang, L., & Jia, Z. (2020). Cost-Effective Resource Provisioning for Deep Learning in the Cloud. *IEEE Transactions on Parallel and Distributed Systems*, 31(7), 1489-1502.

[27] He, X., Song, K., Huang, C., Li, W., & Zhao, L. (2020). Security and Privacy in Federated Machine Learning: Recent Advances and Future Directions. *IEEE Network*, 34(6), 156-165.

[28] Lin, Y., Feng, G., & Zhang, Z. (2020). Machine Learning for Internet of Things Data Analysis: A Survey. *IEEE Communications Surveys & Tutorials*, 22(4), 2543-2590.