MLOps Tools for Deployment: A Case Study on Text Classification

Matea D. Lukić, Danica S. Ivković, Ana M. Poledica

***Abstract*—This paper explores the integration of MLOps tools in the design and operationalization of machine learning pipelines for text classification. Focusing on a case study of web news classification, we examine the use of tools such as MLflow for experiment tracking, Docker for containerization, and Airflow for orchestration. The results reveal both promising advancements and significant limitations, underscoring the challenges of adopting DevOps practices in the rapidly evolving field of machine learning. Although the findings highlight the potential of MLOps to improve scalability and reproducibility, they also demonstrate that the domain is still in its early stages and requires further refinement.**

***Index Terms*—MLOps, deployment, machine learning pipelines, text classification, scalability, reproducibility, operationalization**

I. Introduction

In recent years, machine learning (ML) has evolved into a critical component of modern applications [1], from recommendation systems to the ability to process natural language. However, deploying ML models reliably at scale remains a significant challenge [2], particularly because much of machine learning today still relies on academic datasets and tools that lack scalability for production [3].

MLOps, inspired by DevOps principles, addresses these issues by introducing automation, agility, and collaboration into the development of ML systems [4]. This discipline has emerged as a key to managing the complexities of the development and deployment of machine learning solutions [5]. Unlike traditional software engineering, ML systems pose unique challenges due to their reliance on data, iterative experimentation, and model lifecycle management [6].

This paper provides a comprehensive overview of the deployment tools within the Python ecosystem, and further presents the news classification task as a practical example. It highlights the need for MLOps tools to ensure that solutions are functional in production.

The remainder of the paper is organized as follows. Section II summarizes the findings of the literature review. Section III outlines the development steps and discusses various tool options. Section IV presents a case study on an NLP project. Finally, Section V concludes the paper with a summary and discusses future directions.

M. D. Lukić, Faculty of Organizational Sciences, University of Belgrade, Jove Ilića 154, 11000 Belgrade, Serbia (e-mail: lukicmatea166@gmail.com)

D. S. Ivković, Factory World Wide, Bulevar Mihajla Pupina 115a, 11000 Belgrade, Serbia (e-mail: danica.ivkovic@factoryww.com)

A. M. Poledica, Faculty of Organizational Sciences, University of Belgrade, Jove Ilića 154, 11000 Belgrade, Serbia (e-mail: ana.poledica@fon.bg.ac.rs)

II. Related Work

A. Overview of MLOps

Recent research sheds light on the evolution, components, and practices within the MLOps domain. Platforms like ModelOps, TensorFlow Extended (TFX), and Kubeflow provide end-to-end lifecycle management, orchestrating stages such as data preparation, model training, validation, and deployment into comprehensive ML pipelines [1], [6]. However, challenges remain in resource optimization and efficient handling of pipeline stages. Key points that need to be addressed are bottlenecks such as GPU utilization inefficiencies, which could affect overall system performance [6], and issues like data distribution shifts and model retraining failures [1].

B. Problems in ML System Development

As explained in [7] ML systems often accumulate hidden technical debt, making them fragile and difficult to maintain. A significant issue is entanglement, where tightly coupled components cause small changes to propagate unpredictably, reflecting the CACE principle (Changing Anything Changes Everything). Common antipatterns make these issues even worse, such as glue code, where ad hoc scripts connect disparate system components, and pipeline jungles, complex, undocumented workflows that are hard to debug or reproduce.

C. Case Studies Based on MLOps Paradigm

The study [8] illustrates the effective use of MLOps principles in time-series forecasting. It employs tools such as Docker for containerization, Jenkins for CI/CD, unit tests for validation, PyTorch for model development, MLflow for experiment tracking, and PostgreSQL for data management.

Expanding these ideas [9] integrated tools like Dask, Katib, PyTorch Operator, and KServe within a single Kubeflow Pipelines (KFP) workflow. This setup allowed for anomaly detection using telemetry data from the International Space Station (ISS), showcasing how MLOps can handle real-world challenges by integrating various tools.

These studies provide guidelines for implementing machine learning solutions in complex systems, demonstrating how MLOps principles and a variety of tools can be integrated to address real-world challenges while ensuring reproducibility, scalability, and automation.

III. ML Development Steps

In ML development, as highlighted in [10], finding the right balance between simplicity and flexibility is crucial due to the wide range of available tools, from general-purpose to specialized solutions. The challenge lies in minimizing the complexity of the tool while ensuring adequate functionality. To explore this principle, the development steps [10], [11], are outlined in Fig. 1 and will be explained further.

A. Data Manipulation pipeline

Data manipulation begins with acquiring the necessary data, which ideally already exist in structured repositories. More often, it requires data collection and labeling, using web scraping, APIs or manual collection [4]. Platforms like Kaggle, UCI Machine Learning Repository and Hugging Face Datasets serve as valuable recourses for obtaining curated datasets, while libraries like Scrapy and BeautifulSoup simplify the process of web scraping. For cases where labeled data are needed, tools such as Label Studio, Prodigy, and Doccano assist in efficient annotation.

Data cleaning involves resolving issues such as duplicates, outliers, missing values, and inconsistencies, whereas data transformation involves altering the type or distribution of variables, such as converting numeric values into categorical ones, normalizing the data or even creating new derived variables to enhance model performance [12]. Libraries like Pandas and Polars provide extensive tools for data manipulation.

Data preparation, as outlined in [11], involves organizing the dataset for training and evaluation. It includes splitting the data into training, validation, and test subsets with libraries such as Scikit-learn. The concept of spanning allows for the inclusion of new data snapshots over time, ensuring that the models remain up-to-date without having to retrain from scratch. To maintain reproducibility and consistency, tools like Data Version Control or Pachyderm can be used.

B. Modeling pipeline

Training is the first step in modeling pipline, where a model learns patterns from data by adjusting its parameters [13]. Modern frameworks like TensorFlow, JAX and PyTorch simplify and enhance this process, with PyTorch Lightning providing a streamlined interface for PyTorch. Platforms like Google Colab are essential for providing cloud-based training environments with free GPU/TPU access, making it possible for researchers and practitioners to train computationally intensive models without investing in expensive hardware. Jupyter Notebook and Kaggle Kernels complement this by offering options for both local and cloud-based experimentation, allowing for rapid prototyping, visualization, and interactive debugging. Tools like MLflow, Weights & Biases (Wandb), and Neptune.ai track experiments and ensure reproducibility by recording hyperparameters, metrics, and artifacts.

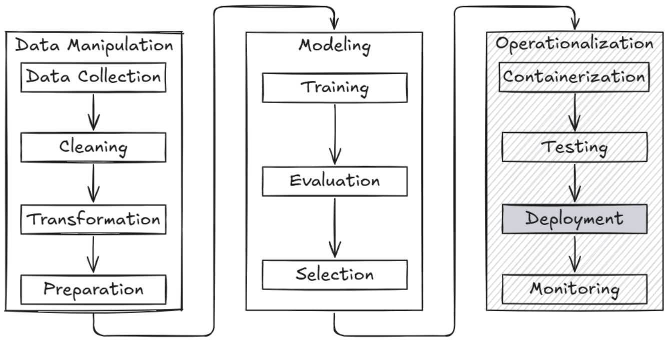

Figure 1. Development steps in ML projects

Evaluation is critical to understanding a model's performance and ensuring its generalizability. Tools like scikit-learn offer a wide range of metrics for evaluating models in classification, regression, and clustering tasks. SciPy and StatsModels provide additional statistical tools for deeper analysis and hypothesis testing. For visualizing and comparing model performance, platforms such as DAGsHub, Guild AI, and Comet.ml are widely used. These tools facilitate identifying overfitting, underfitting, and other performance bottlenecks. The ONNX (Open Neural Network Exchange) format allows evaluation and benchmarking across different frameworks, ensuring interoperability. Additionally, models can be converted to other formats such as H5, ProtoBuf, Pickle, Joblib and TFJS enabling compatibility with specific deployment environments.

Selection involves choosing the best architecture and hyperparameters to achieve the optimal level of flexibility for a model [13]. Tools like Optuna, Ray Tune, and Hyperopt provide automated hyperparameter optimization, leveraging techniques like Bayesian optimization and distributed execution. Experiment management platforms such as MLflow enable tracking and comparing multiple model versions, aiding informed decision-making. For NLP tasks, libraries like Hugging Face (HF) Transformers, spaCy, and AllenNLP offer pre-trained models that simplify selection and customization. Cloud-based platforms like AWS SageMaker, Google AI Platform, and Azure Machine Learning provide managed services for large-scale model selection and deployment.

C. Operationalization pipeline

Containerization is a key concept in modern software development, where applications are packaged in isolated environments called containers. Containers are standalone and executable packages of software that include everything needed to run an application, such as code, system tools, libraries, and settings [14]. While Docker is one of the most widely used tools for this process, Podman offers a daemonless alternative with enhanced security features, such as rootless containers. Kubernetes (k8s), on the other hand, is not a container engine but a container orchestration platform designed to manage and automate the deployment, scaling, and operation of containerized applications across clusters. Unlike Docker and Podman, which focus on creating and running individual containers, Kubernetes operates at a higher level, coordinating the behavior of multiple containers and ensuring high availability and scalability.

The next step is testing, which ensures the reliability and robustness of machine learning pipelines. Libraries such as pytest and unittest support unit testing of pipeline components, while Great Expectations and Deequ validate the quality of input and output data. For integration and system testing, frameworks such as Airflow, Prefect, and Kubeflow Pipelines manage end-to-end workflows, ensuring seamless execution across all stages. Tools like Jenkins provide automated continuous integration and testing pipelines.

Deployment transforms machine learning models into production-ready solutions [3]. Tools like AWS SageMaker, Google AI Platform, and Azure Machine Learning provide managed services for scalable deployment. Open-source frameworks like FastAPI, Flask, and Django facilitate the creation of custom APIs for model inference. Exporting models in formats such as ONNX and TFJS ensures compatibility with edge devices and mobile applications.

Monitoring ensures that the deployed models continue to perform as expected. Tools like Prometheus and Grafana provide real-time metrics and visualization, enabling proactive issue resolution. Platforms like Evidently AI, WhyLabs, and Fiddler monitor data drift (the delta between the changes in the data from the last time the model training occurred [3]), model performance, and fairness, ensuring models remain reliable over time.

IV. Case Study: Text Classification

This case study demonstrates the development and operationalization of a machine learning system designed for daily classification of scraped news using the AG News dataset [15]. The workflow used together with modern tools and frameworks, shown in Fig. 2, ensures robustness and efficiency. Table I summarizes the benefits and alternatives of these tools, while some of their limitations will be discussed later in the text.

A. *Methodology Used*

The initial steps of the pipeline involved data collection and preprocessing. The AG News dataset, sourced from the Kaggle platform [16], was used for training and testing. Once trained, the model was applied to classify news articles, which were scraped daily from various online sources and parsed using BeautifulSoup. Project package management was handled using Poetry.

The data manipulation process involved multiple preprocessing steps to standardize textual data for model training. This included removing HTML tags, converting the text to lowercase for consistency, cleaning the text by eliminating references to news agency names, and combining the title and the news description into a single, coherent text field. Library Polars was used to efficiently handle these tasks.

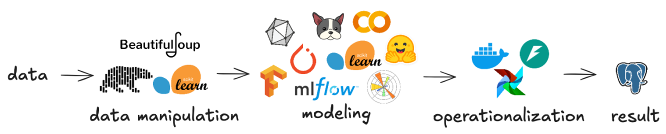

Figure 2. The tools used in each development step

TABLE I
Overview of Tools, Benefits, and Alternatives

Tool	Benefits	Alternatives
BeautifulSoup	Fast, lightweight	Scrappy, Selenium
Polars	Multithreaded, scalable	Pandas, Spark, Dask
Scikit-learn	Documentation, popular	Orange, RapidMiner
PyTorch	Syntax, modern, intuitive	TensorFlow, JAX
Tensorflow	TensorBoard, scalable	PyTorch, JAX
MLFlow	Reproducibility, flexibility	Wandb, Neptune.ai
Google Colab	Fast, Google Drive access	Kaggle, Local build
HF Transformers	SOTA models	AllenNLP, SpaCy
DagsHub	Github integration	Databricks, Wandb
ONNX	Standard model format	H5, TFJS, Joblib
Matplotlib	Most popular	Seaborn, Plotly
Airflow	Wide set of providers	Dagster, Prefect
Docker	Ecosystem, easy to learn	Podman, K8s
FastApi	Less code, performance	Flask, Django
Uvicorn	ASGI server, lightweight	Daphne, Hypercorn
PostgreSQL	Scalable, cheap	SQLite, Casandra

The modeling phase involved training a variety of models using different frameworks and libraries. PyTorch was used to implement Recurrent Neural Networks (RNNs) [17] and Long Short-Term Memory (LSTM) networks [18]. For transformer-based models, Hugging Face's ELECTRA [19] and DistilBERT [20] were utilized, leveraging pre-trained architectures. The text was tokenized with TensorFlow, which, along with other preprocessing steps, prepared the data for model input. Scikit-learn was used to compute the evaluation metrics necessary for comparing the performance of different models. These metrics, including accuracy, precision, recall, and F1-score, were then visualized using Matplotlib to provide clear insight into model performance and guide further improvements. The best results achieved by each model are summarized in Table II, where it is shown that transformer models outperformed traditional recurrent neural network models. For more details on hyperparameters, additional results, and other related information, visit our DAGsHub repository [21].

The combined use of these frameworks facilitated a comprehensive exploration of modeling techniques. Model training was performed in Google Colab, taking advantage of its free GPU and TPU resources to accelerate computations. Experiment tracking, including hyperparameter configurations and performance metrics, was managed through MLflow, which was integrated with DAGsHub to ensure reproducibility and efficient collaboration. The final model was exported in ONNX format, ensuring cross-platform compatibility and optimized inference.

The operationalization of the system was designed to ensure reliability, scalability, and automation. FastAPI exposed the trained model for real-time inference, providing a fast and efficient API endpoint, while Uvicorn served as the asynchronous web server, enabling quick handling of classification requests. Docker was used for containerization, packaging the application along with its dependencies. Task scheduling was orchestrated with Airflow, automating daily scraping and classification processes, which were set to run at midnight. The classification results were stored in PostgreSQL, providing

TABLE II
BEST ACHIEVED PERFORMANCE METRICS FOR EACH MODEL

Model	Accuracy	Recall	F1-score
DistilBERT	0.93598	0.93594	0.93599
ELECTRA	0.92651	0.92669	0.92660
LSTM	0.90867	0.90869	0.90867
RNN	0.87485	0.87468	0.87485

efficient querying and retrieval capabilities for downstream applications.

The visual representation of the architecture, including its components and task flow, is provided in [15].

B. Limitations

Integrating traditional DevOps tools into machine learning workflows presented several challenges. Poetry falls short in addressing specific needs in machine learning workflows, particularly when it comes to specifying the 'CPU-only' version of the Torch library. This limitation complicates workflows when models are trained on platforms like Google Colab rather than on machines with local GPUs.

Apache Airflow, on the other hand, introduced a different set of challenges when deployed in a Docker Compose environment. The official documentation [22] explicitly states that this setup is suitable for learning and experimentation but not recommended for production systems due to the lack of security guarantees. Adapting it for real-world use requires specialized expertise in Docker and Docker Compose, making it less accessible and limiting the support available from the Airflow community.

These examples highlight that MLOps practices are still evolving to meet the demands of machine learning workflows. Although these tools are widely used and well-established in the DevOps world, their integration with machine learning projects requires additional adjustments and workarounds.

V. CONCLUSION

This paper provides a comprehensive overview of MLOps tools and their practical application in the context of text classification, a common NLP problem. The case study on news classification using the AG News dataset highlighted the importance of integrating and automating workflows to enhance model development and deployment. Tools like PyTorch and Hugging Face Transformers were employed for experimenting with different model architectures, while the primary focus was on leveraging MLOps tools such as MLflow for tracking model performance, and Docker and Airflow for containerizing the models and automating the classification pipeline.

The practical application of MLOps in NLP showed their potential to increase the efficiency, reproducibility, and scalability of machine learning workflows. These frameworks enable practitioners to address common challenges, such as managing numerous experiments, handling data distribution shifts, and ensuring real-time monitoring. However, integrating these components into a cohesive MLOps pipeline remains a significant challenge, underscoring that the MLOps field is still in its early stages. As the field evolves, staying informed about emerging tools and best practices will be crucial for effectively managing the complexities of NLP problems and driving continued innovation.

Future iterations of this work could benefit from implementing automated testing throughout the system and continuous model monitoring to track performance over time. Additionally, migrating from local deployment to a cloud-based infrastructure could enhance the system's scalability and reliability while reducing operational overhead.

REFERENCES

[1] Wazir, S., Kashyap, G. S., & Saxena, P. (2023). Mlops: A review. arXiv preprint arXiv:2308.10908.

[2] Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (mlops): Overview, definition, and architecture. IEEE access, 11, 31866-31879.

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

[4] Haviv, Y., & Gift, N. (2023). Implementing MLOps in the Enterprise. " O'Reilly Media, Inc.".

[5] Alla, S., & Adari, S. K. (2021). Beginning MLOps with MLFlow. Apress: New York, NY, USA.

[6] Zhou, Y., Yu, Y., & Ding, B. (2020, October). Towards mlops: A case study of ml pipeline platform. In 2020 International conference on artificial intelligence and computer engineering (ICAICE) (pp. 494-500). IEEE.

[7] 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.

[8] Subramanya, R., Sierla, S., & Vyatkin, V. (2022). From DevOps to MLOps: Overview and application to electricity market forecasting. Applied Sciences, 12(19), 9851.

[9] Steude, H. S., Geier, C., Moddemann, L., Creutzenberg, M., Pfeifer, J., Turk, S., & Niggemann, O. (2024). End-to-end MLOps integration: a case study with ISS telemetry data. In ML4CPS–Machine Learning for Cyber-Physical Systems.

[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] Hapke, H., & Nelson, C. (2020). Building machine learning pipelines. O'Reilly Media.

[12] Brownlee, J. (2020). Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python. Machine Learning Mastery.

[13] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.

[14] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux j, 239(2), 2.

[15] Lukić, M., & Ivković, D. (2024). MLOps-For-NLP [Software]. GitHub. https://github.com/MateaLukiccc/MLOps-For-NLP

[16] Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. Advances in neural information processing systems, 28.

[17] Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. arXiv preprint arXiv:1912.05911.

[18] Hochreiter, S. & Schmidhuber, J. (1997) Long Short-Term Memory. Neural Computation, 9, 1735-1780.

[19] Clark, K., Luong, M.-T., Le, Q. V. & Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. arXiv preprint arXiv:2003.10555.

[20] Sanh, V. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

[21] Lukić, M., & Ivković, D. (2024). MLOps-For-NLP [Software]. DagsHub. https://dagshub.com/MateaLukiccc/MLOps-For-NLP

[22] The Apache Software Foundation. (2024, December 9). Running Airflow in Docker. https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose