ISSN: 2959-6386 (Online), Vol. 2, Issue 3, 2023

Journal of Knowledge Learning and Science Technology

journal homepage: https://jklst.org/index.php/home

Automating Model Deployment: From Training to Production

Amandeep Singla[1], Sandarsh Chavalmane[2]

Abstract

The rapid evolution of machine learning models has led to an increased demand for efficient and streamlined processes in deploying these models from the training phase to production environments. This article explores the critical aspects of automating model deployment, focusing on the seamless transition from the development and training phase to the operational deployment in production settings.

Keywords: Automated Model Deployment, Model Training, Production Environment, Continuous Integration and Continuous Deployment (CI/CD), Containerization, Orchestration Tools

.

Article Information:
Article history: *Received:*04/04/2023 *Accepted:*13/04/2023 *Online:* 14/04/2023 *Published*: 14/04/2023
DOI: https://doi.org/10.60087/jklst.vol2.n3.p347
[i] **Correspondence author:** Amandeep Singla

Introduction

Introduction:
Since the technology's birth, deploying machine learning models in a production setting has proven to be a constant difficulty. Professionals from a variety of fields, such as data scientists, machine learning engineers, front-end engineers, and production engineers, have struggled with the difficulties of teamwork in deploying models that are ready for usage in production throughout the years. Only a small percentage of machine learning programs successfully reach the production stage because, despite concentrated efforts, resolving the many challenges involved with this task remains elusive.
A collection of methods and resources has developed in response to these difficulties in order to deal with the complexity of the machine learning lifecycle. Processes like data preprocessing, model building, training, assessment, deployment, and monitoring are intended to be made more efficient by these technologies. The toolkit keeps changing as artificial intelligence (AI) research and development progress. MLOps, or machine learning operations, is essentially the standardization and optimization of the administration of the machine learning lifecycle.
But a closer look is necessary to understand why the machine learning lifecycle needs to be so carefully streamlined. Transferring a business challenge to a high-level machine learning model seems simple at first glance. However, the creation and implementation of numerous machine learning models in a real-world setting is still a relatively new undertaking for a lot of conventional firms. Up until recently, management was made easier by the manageable size of the models or the minimal interest in understanding these models on an enterprise-wide scale.
The increasing prevalence of decision automation, which involves making decisions primarily without human involvement, highlights the need of models. At the same time, organizational risk management for these models becomes more important. MLOps is unique in that it plays a critical part in scaling machine learning initiatives to a scale where significant business value is realized, in addition to its function in reducing the risks associated with

machine learning models in production.

MLOps discipline is required when scaling from a small number of models to tens, hundreds, or thousands that favorably impact business outcomes. In order to fully understand MLOps, one must have a thorough understanding of machine learning and all of its complexities. Although algorithm selection is frequently disregarded in the context of MLOps, it has a direct impact on MLOps operations and is crucial to the construction of machine learning models. At its core, machine learning is the study of computer algorithms that, instead of requiring explicit programming, learn from experience on their own. These algorithms build a software model that can make predictions by analyzing training data.

Model Development

1. Determining Business Goals

A business aim, which might be as straightforward as lowering the percentage of fraudulent transactions to 0.1% or being able to identify faces in social media images, is usually the first step in the creation of a machine learning model. Business objectives, by definition, comprise cost limitations, performance targets, and needs for technical infrastructure. All of these elements can be recorded as key performance indicators, or KPIs, which make it possible to track the business performance of models that are being used in production. It is essential to realize that machine learning projects are usually a subset of bigger initiatives that impact people, processes, and technologies. Therefore, defining goals involves change management, which could offer some direction for developing the ML model. For instance, the necessary degree of openness will have a significant impact on algorithm selection and might necessitate the inclusion of explanations in addition to predictions in order to convert predictions into wise business decisions. Different MLOps Challenges, Different Ml Algorithms All machine learning algorithms rely on modeling patterns in historical data to draw conclusions; the accuracy and applicability of these models play a crucial role in the algorithms' performance. They vary in that every kind of algorithm has distinct qualities and poses different difficulties in MLOps. Algorithm selection may also be influenced by governance issues, even though some ML algorithms are better suited for particular use cases[1]. Opaque algorithms like neural networks are not allowed in highly regulated environments (financial services, for example), where judgments have to be justified. Decision trees and other simpler methods are required. Cost is frequently the trade-off rather than performance in many situations. Put differently, less sophisticated approaches usually require more costly human feature engineering to get the same degree of efficiency as more sophisticated ones.

2. Sources of Data and Investigative Data Analysis

It's time to gather data scientists and subject matter experts to start building the ML model now that there are defined business objectives in place[2]. The first step in this process is to look for relevant input data. Although it can seem like a straightforward operation, actually finding the data can be the most challenging aspect of the process. It is always advantageous to comprehend the patterns in data before attempting to train models, as ML methods are dependent on data[3]. Techniques for exploratory data analysis (EDA) can help with the process of choosing potentially important features, identifying data cleaning requirements, and developing hypotheses about the data. EDA can be carried out statistically for more rigor or graphically for intuitive insight.

Data Exploration

Data scientists and analysts must first comprehend the nature of the data before considering data sources for model training. A model, even one trained by an algorithm, is only as good as the data it uses for training. Many problems, including inconsistency, correctness, incompleteness, and so on, may make any or all of the data unusable at this point. These procedures can include, for instance:

• Examining the statistics that summarize the data and recording the methods used to gather the data and any presumptions that were made: Which column's domain is it? Do any rows have any missing values? Glaring errors? Unusual deviations? Not a single outlier? Analyzing the data distribution more closely; cleaning, filling, reshaping, filtering, clipping, sampling, etc.

• Fitting distribution curves, performing statistical tests on certain subpopulations, and examining correlations between the various columns

• Analyzing that data in relation to other models or data found in literature: Exists any customary information that appears to be absent? Is the distribution of this data comparable? Naturally, in order to make wise decisions throughout this exploration, subject expertise is needed. Certain abnormalities could be challenging to spot without specialized knowledge, and assumptions can lead to unexpected outcomes for the inexperienced eye. Industrial sensor data is a prime example. A data scientist may not know what is normal versus unusual outliers for a particular machine unless they are also an expert in mechanical engineering or equipment.

3. Feature Engineering and Selection

Feature engineering and feature selection are typically the results of EDA. Feature engineering is the act of turning unprocessed data from chosen datasets into "features" that more accurately depict the core issue that needs to be resolved. "Features" consist of fixed-size arrays of numbers as machine learning algorithms can only comprehend them. A large portion of an ML project's effort can be attributed to the feature engineering step of data purification.

● Techniques for Feature Engineering One-hot encoding is the most popular[4]. In contrast, inputs that are text or image-based require more complex engineering. Recent advances in deep learning have revolutionized this subject by producing models that translate text and visual data into tables of numbers that machine learning algorithms can exploit. Known as embeddings, these tables allow data scientists to do transfer learning because they may be applied to domains in which they haven't been trained before.

● Selection of Features The question of how much and when to quit is a common one in feature design and selection. Increasing the number of features could lead to improved model accuracy, more equitable group division, or make up for other valuable missing data. It does, however, have several disadvantages, all of which could eventually have a big effect on MLOps plans. Heuristics are used in automated feature selection to determine the significance of a given feature on the predictive performance of the model. For instance, one can rapidly train a basic model on a representative portion of the data and then look at which features are the best predictors, or one can look at the correlation with the goal variable.

4. Training and Evaluation

Training comes next, following feature engineering and selection for data preparation. Iterative steps are involved in training and optimizing a new machine learning model: testing many algorithms, automatically generating features, adapting feature selections, and fine-tuning algorithm hyperparameters. Apart from its iterative character, training is the most computationally demanding stage in the life cycle of an ML model. In fact, this is the case in many scenarios. It gets harder and harder to remember the outcomes of each experiment when you iterate. The most frustrating thing for data scientists is when they can't remember the precise setup and so can't replicate the optimal outcomes. Remembering the data, choosing features and model parameters, and keeping track of performance indicators may all be made much easier using an experiment tracking tool. They make it possible to compare experiments side by side and identify variations in performance. Selecting the optimal solution requires taking into account both qualitative and quantitative factors, such as the algorithm's explainability or ease of deployment, as well as quantitative factors like accuracy or average error[3].

● Trial and error

Throughout the whole process of developing a model, experiments are conducted, and most significant decisions or assumptions are supported by at least one experiment or body of prior research.Data scientists must be able to swiftly go over every option for each model when doing experiments. Fortunately, all of this can be done semi-automatically with the help of tools; all that's required is defining what has to be tested (the space of possibilities) based on prior information (i.e., what makes sense) and restrictions (i.e., computation, budget). It quickly becomes impossible to try every possible combination of hyper parameter, feature processing, etc. As a result, it's helpful to establish a budget for time and/or computation for trials as well as an acceptable cutoff point for the model's utility (more on that idea in the following section). Thankfully, an increasing number of platforms for data science and machine learning make it possible to automate these workflows not just for the initial run but also for repeatability by preserving all processing actions. Some also permit the testing of theories through the use of experimental branch spin-offs and version control, which can subsequently be combined, abandoned, or retained.

● Assessing and Comparing Frameworks

A British statistician from the 20th century named George E. P. Box famously stated that while all models are incorrect, some can be helpful. To put it another way, a model shouldn't strive to be flawless; instead, it should meet the criteria of being “good enough to be useful” while monitoring the uncanny valley, which is usually caused by models that appear to be performing well but actually perform poorly or disastrously for a particular subset of cases (such as an underrepresented population). In light of this, it's critical to assess a model within its historical context and possess the capacity to draw comparisons between it and prior models or rule-based processes in order to have a sense of the potential consequences of substituting the new model for the present model or decision-making process.

Selecting Evaluation Measures Selecting the appropriate measure to assess and contrast various models for a certain issue may result in drastically different models. For instance, precision is frequently employed in automated classification tasks, however it is seldom the optimal choice when the classes are imbalanced, meaning that one of the possible outcomes is more improbable in comparison to the other. A model that consistently predicts the negative class is therefore 95% correct, but completely useless, in a binary classification problem where the positive class (i.e., the one that is useful to forecast because its prediction initiates an action) is rare, say 5% of occurrences.

Cross-testing, which involves evaluating a metric on a portion of the data that was not used for the model's training (a holdout dataset) might give an indication of how well a model will generalize. Metric evaluation and hyperparameter optimization are two examples of processes where some data is retained for assessment and the remaining portion is used for training or optimization. There are other approaches as well; it's not always a straightforward divide. For instance, data scientists rotate the portions they hold out to analyze and train several times in k-fold cross-validation. This increases the training time but provides a sense of the metric's stability. Data scientists frequently wish to retrain models on more recent data using the same features, hyperparameters, algorithms, and other elements. Obviously, comparing the two models and evaluating the performance of the updated version comes next. However, it's also critical to confirm that all prior hypotheses remain valid, such as that the problem hasn't changed much and that the earlier modeling decisions still make sense given the available facts. More precisely, this is a component of drift and performance monitoring. Regretfully, there isn't a metric that works for everyone. Selecting the appropriate one for the task at hand necessitates knowing the trade-offs and limitations of the measure (the mathematical side) and how they affect the model's optimization (the business side).

5. Version Management and Reproducibility

When evaluating and comparing models for a variety of reasons, including fairness as was just discussed, version control and reproducibility of various model versions are often brought up. Data scientists must be able to maintain consistency across all of the model iterations as they develop, test, and refine them. Reproducibility and version control meet two distinct needs:

● Data scientists may find themselves experimenting, making a lot of various judgments, trying out new combinations, and then making a change when the expected results are not achieved. This entails being able to return to earlier iterations of the experiments—for instance, bringing a project back to its original state when the experimentation process reached a dead end.

● Several years after the first experimentation, data scientists or other stakeholders (auditors, managers, etc.) might want the ability to reproduce the calculations that resulted in the model deployment for an audit team.

● Reproducibility Even though most trials are short-lived, important iterations of a model need to be preserved for later use. Reproducibility, a fundamental idea in experimental science generally, is the topic at hand. Saving enough data about the environment in which the model was generated to enable its creation from scratch with identical outcomes is the aim of machine learning. Data scientists will find it difficult to confidently iterate on models without reproducibility, and they will have even less chance of transferring the model to DevOps to test if what was produced in the lab can be properly recreated in production. A record of the software environment and version control over all the parameters and assets—including training and assessment data—are essential for achieving true repeatability.

6. Productionalization and Deployment

A key element of MLOps is model productionalization and deployment, which poses an entirely other set of technological difficulties from model development[5]. The DevOps team and software engineers are in charge of it, and it is important to recognize the organizational difficulties in handling information sharing between data scientists and these groups. Failures or delays in deployment are inevitable in the absence of efficient teamwork. Types and Contents of Model Deployment Take a step back and consider the following questions to better understand what goes into production and what makes up a model: what goes into manufacturing exactly? Two categories are often used to categorize model deployment:

● Model-as-a-service or live-scoring model Typically, the model is implemented in a straightforward framework to offer a real-time REST API endpoint that allows the API to access the resources needed to complete the task.

● Integrated model In this instance, the model is put together as an application that is released. An application that offers batch-scoring of requests is a typical example. Depending on the technology being utilized, the components of to-be-deployed models can vary, but they usually consist of a collection of data artifacts and code (usually in the form of Python, R, or Java) [6]. Because using different versions may result in different model predictions, any of these may have runtime and package version dependencies that need to match in the production environment. Model export to a portable format (PMML, PFA, ONNX, or POJO) is one way to reduce reliance on the production environment. These are designed to make deployment easier and promote model portability between systems. But they have a price: only a small number of algorithms are supported by each format, and occasionally the portable models perform slightly differently from the original. Making the decision to employ a portable format requires a deep comprehension of the business and technological environment[7]

● Containerization

When deploying ML models, containerization is gaining popularity as a solution to dependencies-related issues.

Virtual machines can be replaced by lightweight container technologies like Docker, which enable the deployment of applications in separate, self-contained environments that are customized to meet the unique requirements of each model.They also make it possible to use the blue-green deployment technique to easily introduce new models. Multiple containers can also be used to elastically scale model compute resources. Multiple containers can be orchestrated by using on-premises and cloud-based technologies like Kubernetes.[8]

● Environments at Runtime

Verifying that a model can be produced technically is the first step before submitting it to production. perfect Rapid, automated deployment is preferred by MLOps systems over labor-intensive procedures, and the approach that works best can be greatly influenced by runtime settings. Production environments come in many different shapes and sizes: JVMs running on embedded systems, Kubernetes clusters, specialized services like TensorFlow Serving, custom-built services, data science platforms, etc. The fact that various diverse production environments coexist in some organizations adds to the complexity of the situation.[9]

Models that are operational in the development environment should ideally be validated and transferred to production exactly as is. This reduces the amount of work required for adaptation and increases the likelihood that the production model will behave similarly to the development model. Sadly, this ideal situation is not always achievable, and teams have been known to complete a lengthy project just to discover it cannot be implemented.

Conversion of Development Environments to Production Environments When it comes to adaptation effort, on the one hand, the dev model can operate in production without any changes if the development and production platforms are from the same vendor or are otherwise interoperable. In this instance, a few clicks or instructions suffice to complete the technical procedures needed to send the model into production, freeing up all work to be directed toward validation. Conversely, there are situations in which a new implementation of the model is required, perhaps by a different team and using a different programming language. There aren't many situations these days where this strategy makes sense, given the costs and time involved. But it's still a reality in a lot of organizations, and it usually results from a lack of suitable procedures and tools. In actuality, a model won't enter production for months or even years if it is turned over to another team to reimplement and modify for the production environment. To make the model production compatible, a variety of changes can be made to it or its interactions with the environment in between these two extreme scenarios. Validation should never take place in the development environment; instead, it should always take place in an environment that is as similar to production as possible.

● Tool-related factors Early consideration should be given to the format that must be used for submission to production, as this can significantly affect the model itself and the amount of effort needed to productionalize it. For instance, conversion is plainly needed when a model is constructed in a Python environment using scikit-learn and the production environment is Java-based and requires PMML or ONNX as input.

● Performance-related factors Performance is another frequent reason that conversion could be necessary. For instance, a Python model converted to C++ will usually have a lower scoring latency than its equivalent in Python. Although it certainly relies on many circumstances, the final model may be dozens of times slower, the likelihood is that it will be dozens of times quicker.

● Data Access Prior to Production Launch and Validation Data may occasionally be frozen and combined with the model. However, in situations where this isn't feasible (like when the dataset is too big or enrichment data must always be current), the production environment should be able to access a database. To do this, it needs to have the necessary network connectivity, installed libraries or drivers, authentication credentials saved in a production configuration, and the necessary libraries or drivers. In reality, managing this setup and configuration may be very difficult because, once again, it calls for the right tools and teamwork (especially when scaling to several dozen models). Model validation is especially more important when using external data access in production-like scenarios since production malfunctions are frequently caused by poor technological connectivity.

7. Model Deployment Requirements

What therefore needs to be taken care of throughout the productionalization phase, which occurs between the conclusion of model creation and the actual deployment into production? There's no denying that quick, automated deployments are always better than laborious ones. In many cases, self-service applications with limited lifespans don't need testing or validation. Fully automated single-step push-to-production might be more than sufficient if technologies such as Linux groups can properly cap the highest resource demands of the model [10]. With frameworks like Flask, handling numerous user interfaces is even doable while utilizing this lightweight deployment strategy. Apart from platforms that combine data science and machine learning, certain business rule management solutions might enable the automated implementation of fundamental machine learning models. A more comprehensive CI/CD pipeline is needed in use cases that involve customers and mission-critical operations. Usually, this includes:

Verifying that all requirements for coding, documentation, and sign-off have been fulfilled
2. Replicate the model in a setting that resembles a production setting.
3. Verifying the model's accuracy again
4. Carrying out explainability analyses
5. Verifying that all governance criteria have been fulfilled
6. Evaluating any data artifacts for quality
7. Examining how resources are used when under stress
8. Integrating, together with integration tests, into a more intricate application

8. Monitoring

A model must maintain good performance over time when it is put into production. However, various people have varied definitions of what constitutes good performance; this is especially true for the DevOps team, data scientists, and business.[11]

● Issues with DevOps

The well-known concerns of the DevOps team include things like: 1. Is the model doing the task rapidly enough?
2. Is the memory and processing time being used appropriately? DevOps teams are accustomed to performing traditional IT performance monitoring in this manner. In this sense, ML models' resource needs are comparable to those of conventional software.

It's crucial to take computing resource scalability into account. All things considered, using the knowledge now possessed by DevOps teams for resource management and monitoring to ML models is simple.

● Concerns of Data Scientists

The actual world never stops. A new kind of fraud that emerged in the previous three months will not be reflected in the training data that was used to create a fraud detection model six months ago. A model that generates advertisements is likely to create fewer and fewer appropriate advertisements as a website starts to draw in younger users. The performance will eventually reach an unacceptable level, requiring retraining of the model. How fast the real world changes and how accurate the model needs to be, but most crucially, how easy it is to create and implement an improved model, will dictate how often models need to be retrained.[12]

● The actual truth

In short, the ground truth is the right response to the query that the model was trained to answer.By obtaining the ground truth for each prediction a model has produced, one can assess the model's effectiveness. Sometimes a prediction is followed quickly by ground truth.

● Drift in input

The theory behind input drift holds that a model can only make accurate predictions if the training data accurately depicts the real world. Hence, the model performance is probably affected if recent queries to a deployed model are compared to training data and show notable discrepancies. This serves as the basis for tracking input drift. The best thing about this strategy is that it doesn't require waiting for additional information or ground truth because all the data required for the test is already available. One of the most crucial elements of a flexible MLOps approach is recognizing drift, which can also increase the organization's overall enterprise AI efforts' adaptability.

● Life Cycle and Iteration

A crucial and challenging step in the MLOps life cycle is creating and implementing upgraded models. Model performance decline as a result of model drift is one of the reasons for creating a new version of the model, as was covered in the section before this one. Sometimes the data scientists have just improved the model's design, and other times there is a need to take into account more precise business objectives and KPIs. Repetition Every day, new training data is made available in certain industries that experience rapid change. The model is regularly automated for daily retraining and redeployment to guarantee that it appropriately represents current experience. Retraining a current model using the most recent training data is the simplest way to iterate a new model version. Nevertheless, there are still a lot of traps even with the feature selection and process remaining same. More particular still:

● Does the newly acquired training data seem as anticipated? It is crucial to automatically validate the new data using pre-established metrics and checks.

● Is the information consistent and full?

● Do the feature distributions resemble those from the prior training set in general? Recall that the objective is to improve the model, not to drastically alter it. Examine the metrics between the newly created model version and the active model version. In order to accomplish this, the models need to be assessed using the same development

dataset—regardless of its most recent iteration. It is best to seek out manual intervention and avoid reusing automated scripts if measurements and checks reveal a major difference between the models.

● Multiple development datasets based on scoring data reconciliation (with ground truth when it becomes available), data cleaning and validation, the prior model version, and a set of carefully considered checks are required, even in the "simple" automated retraining scenario with new training data. Automated redeployment is improbable because retraining in other contexts is probably considerably more difficult.

In large businesses, the live model scoring environment and the model retraining environment should typically be kept apart according to Feedback Loop DevOps best practices. It is therefore quite likely that a new model version will be evaluated in the retraining environment. Shadow testing, in which the updated model version is introduced into the actual environment alongside the current model, is one method of reducing this uncertainty. The current model version manages all live scoring; however, every new request is scored anew by the new model version, with the results logged but not given back to the requester. The outcomes can be statistically compared once both versions have scored a sufficient number of requests. Additionally, shadow scoring increases the SMEs' visibility on upcoming model iterations, which could facilitate a more seamless transition.

Conclusion:

For many organizations, putting machine learning models into production is a major challenge. MLOps is the cornerstone for ensuring deployed models are well-maintained, function as planned, and don't negatively impact the business as AI initiatives grow. As a result, using appropriate MLOps techniques is crucial. A machine learning (ML) model cannot realistically operate in production for years without updates, unlike normal software. Model predictions have an intrinsic deterioration that necessitates frequent retraining. Handling these updates by hand gets tiresome fast and is not scalable. The first step in automating processes is figuring out which metrics to track, when these data start to cause concern, and which indications are utilized to assess if a new model iteration is outperforming the old one. These difficulties emphasize how crucial it is to view MLOps as a whole, with the components originating from model design, construction, deployment, monitoring, and governance. We have spoken about how to address some of these problems in the development of contemporary machine learning systems in this work.

Reference List:

1. (2023). Towards a safe MLOps Process for the Continuous Development and Safety Assurance of ML-based Systems in the Railway Domain. doi: 10.48550/arxiv.2307.02867

2. Ayesha, Tabassam. (2023). MLOps: A Step Forward to Enterprise Machine Learning. arXiv.org, doi: 10.48550/arXiv.2305.19298

3. Boris, Bertolt, von, Siandje. (2023). MLOps: A Step Forward to Enterprise Machine Learning. doi: 10.48550/arxiv.2305.19298

4. Sibanjan, Das., Pradip, Kumar, Bala. (2023). What drives MLOps adoption? An analysis using the TOE framework. Journal of Decision Systems, doi: 10.1080/12460125.2023.2214306

5. Lincoln, Costa. (2023). An investigation of challenges in the machine learning lifecycle and the importance of MLOps: A survey. Anais do Computer on the Beach, doi: 10.14210/cotb.v14.p379-386

6. A. Singla, D. Sharma and S. Vashisth, "Data connectivity in flights using visible light communication," 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), Gurgaon, India, 2017, pp. 71-74, doi: https://doi.org/10.1109/IC3TSN.2017.8284453

7. F. Lin et al., "Predicting Remediations for Hardware Failures in Large-Scale Datacenters," 2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S), Valencia, Spain, 2020, pp. 13-16, doi: https://doi.org/10.1109/DSN-S50200.2020.00016

8. N. Sullhan and T. Singh, "Blended services & enabling seamless lifestyle," 2007 International Conference on IP Multimedia Subsystem Architecture and Applications, Bangalore, India, 2007, pp. 1-5, doi: https://doi.org/10.1109/IMSAA.2007.4559085

9. Building for scale. (n.d.). https://scholar.google.com/citations?view_op=view_citation&hl=en&user=jwV-mi8AAAAJ&citation_for_view=jwV-mi8AAAAJ:zYLM7Y9cAGgC

10. Wu, K. M., & Chen, J. (2023). Cargo operations of Express Air. *Engineering Advances*, *3*(4), 337–341.

https://doi.org/10.26855/ea.2023.08.012

11. Wu, K. (2023). Creating panoramic images using ORB feature detection and RANSAC-based image alignment. *Advances in Computer and Communication*, *4*(4), 220–224. https://doi.org/10.26855/acc.2023.08.002

12. Liu, S., Wu, K., Jiang, C. X., Huang, B., & Ma, D. (2023). Financial Time-Series Forecasting: towards synergizing performance and interpretability within a hybrid machine learning approach. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2401.00534