h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Darmstadt University of Applied Sciences

Faculty of Mathematics and Natural Sciences & Informatics

Tracking Signal Usage for Time Series Forecast Monitoring in an AWS MLOps Environment

Submitted in partial fulfilment of the requirements for the degree of

Master of Science (M.Sc.)

in the course of studies Data Science

by

Gerrit Derk Scheppat

Matriculation number: 769050

First Examiner:	Prof. Dr. Christoph Busch
Second Examiner:	Prof. Dr. Jutta Groos

Issue Date:	01.05.2022
Submission Date:	15.08.2022

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 15. August 2022

Gerrit Derk Scheppat

ABSTRACT

Machine Learning Operations (MLOps), the task of coordinating machine learning projects with multiple models and team members, is growing in importance and interest. Cloud computing resources are a popular option in this scenario due to many reasons like easily accessible computing resources, billing by usage time and further available services like a fully managed environment. Two approaches to monitor models in an MLOps environment are compared by using two popular statistical time series forecasting models and two datasets from a widely known forecasting competition. One approach is the default Amazon Web Services (AWS) model drift monitoring and the other is a tracking signal monitoring. The goal is to reduce economical and ecological costs generated by retraining deployed models with more recent data in a cloud environment. The tracking signal monitoring is shown to serve as a more generic approach which can reduce costs when a decreased model performance is accepted for lower training costs. The AWS monitoring with in-sample error metrics as monitoring threshold used as retraining trigger shows a better performance at a comparable level of retraining counts.

ZUSAMMENFASSUNG

Projekte des maschinellen Lernens mit mehreren Modellen und Teammitgliedern zu koordinieren (MLOps) ist eine Aufgabe wachsender Relevanz. Aus verschiedenen Gründen wie dem leichten Zugang zu Rechenressourcen und weiteren angebotenen Leistungen wie einer vollständig verwalteten Rechenumgebung sind Cloud Computing Ressourcen eine beliebte Option in diesem Szenario. Die Abrechnung ist dabei meist an die Nutzungszeit gekoppelt. Um zu evaluieren, wie sich die Kosten in einer Cloud Computing MLOps Umgebung in Verbindung mit Modellen aus dem Bereich der Zeitreihenvorhersage abhängig von der Überwachungsvariante verhalten, werden verschiedene Herangehensweisen verglichen. Dabei sollen ökonomische und ökologische Kosten, die durch das Nachtrainieren von Modellen mit neueren Daten entstehen, reduziert werden. Dazu werden zwei beliebte statistische Modelle und zwei Datensätze aus einem bekannten zeitreihenprognose Wettbewerb evaluiert. Als Überwachungsmethode wird die standardmäßige Überwachungsvariante für MLOps Umgebungen von Amazon Web Services (AWS) mit einer Tracking Signal gestützten Überwachung verglichen. Die Tracking Signal Überwachung bietet einen generischeren Ansatz, welcher Kosten reduzieren kann, wenn eine reduzierte Vorhersagekraft von Modellen für verringerte Kosten durch das Nachtrainieren akzeptiert werden kann. Die AWS Überwachung, welche die Fehlermetriken aus den Trainingsdaten als Grenzwert nutzt um ein Nachtrainieren auszulösen, zeigt eine bessere Leistung bei einem vergleichbaren Level von Nachtrainierungsiterationen.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI Artificial Intelligence

AIC Akaike's Information Criterion

AR Autoregressive

ARIMA Autoregressive Integrated Moving Average

AWS Amazon Web Services

BIC Bayes Information Criterion

CD Continuous Deployment

CI Continuous Integration

DevOps Development & Operations

ETS Error, Trend, Seasonal

LSTM Long Short-Term Memory

MA Moving Average

MAE Mean Absolute Error

MAPE Mean Absolute Percent Forecast Error

ML Machine Learning

MLOps Machine Learning Operations

MSE Mean Squared Error

On-Prem On Premises

RMSE Root Mean Squared Error

ROC Reciever Operating Characteristic

SMAPE Symmetrical Mean Absolute Percentage Error

Part I

THESIS

1 INTRODUCTION

1.1 MOTIVATION AND APPLICATION CONTEXT

Mathematical models generated with statistical tools or Machine Learning (ML) algorithms have prominently arrived in society. The usage of Artificial Intelligence (AI) is discussed on a political scale, evaluating current trends, opportunities and risks. On a european level, it is acknowledged that AI can generate improvements in different sectors like healthcare, farming, climate change, production efficiency or security although risks involved with AI like opaque decision making, discrimination or criminal purposes need to be observed. A shift to edge computing [Cao+20] and more industrial data is expected but currently, cloud-based processing of consumer related data is dominating. On this political level, policy options and objectives are discussed to succeed in the international race for AI leadership aiming e.g. for legal certainty, enforcement of existing laws and fundamental rights, accountability, technical robustness and safety [Com20; Com21]. With progress in big data and deep learning, the usage of AI and ML is present in various fields [CML14; LBH15]. Typical applications include producing recommendations [Zha+19], giving various kinds of forecasts [HLV16; AC20; SCH20], processing images [O'M+20] and text [MRS08] or grouping and labeling entities [Oye+19]. The increasing usage of those models often requires life cycle management to support the organization between the different roles involved in producing, hosting and maintaining models which includes data scientists, data engineers, software architects and Development & Operations (DevOps) specialists. Besides the amount of different roles involved, the typically required large datasets for model training bring additional challenges with data storage and management [Agr+19; Den+09]. These challenges lead to the interest in Machine Learning Operations (MLOps) architectures that provide functions to organize the workflow connected to the model life cycle. There are several solutions available providing MLOps services and as one of the major actors in cloud computing, Amazon Web Services (AWS) provides MLOps tools as well. Those are implemented in their fully managed environment and target a quick setup for the customer with minimal engineering requirements. Especially time series forecast models need to be monitored after deployment and retrained with new data since prediction accuracy can quickly degrade when the underlying patterns that the original model was trained on, change [DJI04]. Therefore, the special requirements of time series forecast models in an MLOps environment need to be understood and solutions need to be implemented to keep the model prediction accuracy and the costs for maintaining the models on an efficient level.

1.2 GOAL OF THIS THESIS

The possibilities to use AWS as an MLOps environment with focus on time series forecasting are examined and possible weaknesses in the default monitoring of model drift are investigated. As an alternative solution, monitoring with a tracking signal is conducted. Since the usage of the MLOps tools at AWS are connected to costs, the goal of this thesis is to identify, which monitoring method results in minimal costs regarding the costs of retraining, the costs of degraded model performance in a business case and the amount of CO_2 generated while retraining models depending on the different monitoring options. To achieve this goal, two datasets with ten time series each are monitored over 100 forecasting data points. Three different error measures (Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE)) are evaluated over this time span with each two statistical models (Autoregressive Integrated Moving Average (ARIMA), Error, Trend, Seasonal (ETS)) and two monitoring options (AWS model drift, tracking signal) resulting in different amounts of model retraining times. To benchmark these results, two more options to retrain the models are evaluated: never retrain the model and retrain the model after every one-step-ahead prediction with the latest data point. Two public time series datasets with different characteristics are used. To compare the performance of the two monitoring approaches at the same retraining count, an adapted tracking signal is examined additionally, which results in the same retraining count level as the AWS monitoring method.

MLOps frameworks rely on automation and reusability. Monitoring methods and thresholds should follow these principles. The customization of AWS monitoring thresholds is very specific to the data and the used models. Therefore, to adapt the level of the targeted retraining count, knowledge about the data and the performance of the forecasting model is required. To assess if the tracking signal monitoring can serve as a more generic monitoring method then the AWS approach, it is evaluated if a specific threshold of the tracking signal results in comparable retraining counts over multiple datasets and models. If a tracking signal can be adjusted to a specific level and a retraining count can be expected from that threshold independent of the data or model, the operation with tracking signals would be more universal.

With the results of this research, guidance can be given on how to decide on a time series forecasting MLOps system and the best way on monitoring and retraining scheduling can be implemented in a cost efficient manner.

1.3 STRUCTURE

After this introduction to the topic and the goal of the research, chapter 2 gives a more detailed introduction to the theoretical background regarding MLOps, time series forecasting including the characteristic data, prominent and state-of-the-art models, different monitoring metrics and different kinds

of possible drifts which can degrade the prediction accuracy of deployed models. Chapter 3 explains the MLOps solutions provided by AWS and connects the described goal of this thesis with possible weaknesses found in the AWS framework by introducing the research hypothesis. Chapter 4 summarizes related work in the time series forecasting monitoring field. In chapter 5 the datasets, the experiment framework and the experiment results are described. Chapter 6 discusses the results of the two datasets in relation to the research hypothesis. Finally, chapter 7 gives a conclusion of the research and describes limitations and possible further research questions and methods.

2

THEORETICAL BACKGROUND

2.1 ML OPS

To understand why there is a need for MLOps and to understand what MLOps is, the typical workflow of a ML project needs to be understood. The first step towards a model which performs well in a real life scenario is the business or research problem. For every problem, an evaluation should be conducted on which kind of solution is a good approach for this specific problem. If the answer is that a ML model in production is a good solution, the workflow most machine learning specialists or data scientists know (fig. 2.1), begins.

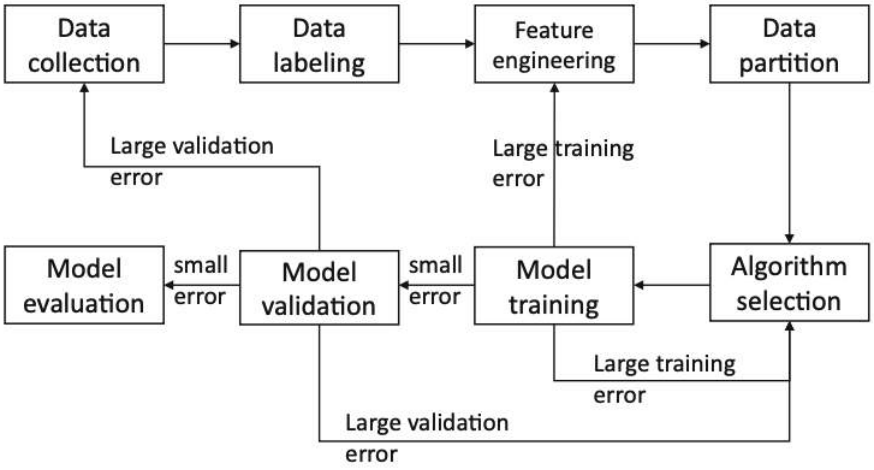


Figure 2.1: Typical ML Workflow [Qin20]

Every ML and statistical model requires data to learn from and adjust model parameters to. Therefore data relevant to the problem must be collected and persisted for further engineering processes. Which data can be considered as relevant is often not clear and also depends on the desired usecase. The decision which data is considered relevant can be made e.g. by data scientist, project leads or other stakeholders. Data can be already stored but not sufficiently preprocessed or entirely new data has to be collected. In this collection step, data stays in its raw format and just a very rough differentiation between relevant and non relevant data is made, in doubt all candidates for further processing can be persisted if storage capacity permits. At this step it is crucial to have sufficient variance in the data especially in the target value so that for example in a model which should forecast if it will rain tomorrow there are days with rain and days without rain in the dataset. For non target values this should also be taken into con-

sideration. The data collection step can also include a first processing where deviant data like null values or faulty data is dropped [AA20].

For supervised ML problems, a data labeling process follows where, mostly manually, ground truth labels are added to the data. At the feature engineering step, several possible operations to split, merge or transform data enable the model to access the data. When the data is ready for usage, in data partitioning, it is typically split into train, validation and test data with different proportions to open the possibility of recognizing when a model overfits. The goal is to avoid that a model is adjusted very well to known data but performs much worse on unknown data. Depending on the given circumstances, the training data split usually gets the biggest share of data and is used to adjust the parameters or weights of a model to perform well on this data. To make sure the model does not only work well on the training data, validation data is used to validate the performance of a model on unseen data and to experiment with different hyperparameters. The final assessment of the model performance is conducted on the test data in the model evaluation. In the iterative process of going back to feature engineering, change the model algorithm or hyperparameters or even the need to collect more or different data, a model which performs well on unseen data is constructed.

Up to this step the classic data science team might be familiar with the workflow but the life cycle of a model does not stop at this point, the responsibility just transfers to engineering and operational teams to deploy and monitor the model. The model has to be integrated into applications and a production environment to benefit the business case [AA20]. The model has to be constantly monitored to assure the performance does not degrade over time with changing data or user behaviour. When a degrading model performance is monitored, the whole process of the ML workflow has to begin again (fig. 2.2). The monitoring of external factors like changing data or user behaviour is one aspect why the model might require retraining or adjustment. Another reason for change is when business requirements change and the model does not serve as the desired solution anymore.

It should be clear that this process is very time consuming in each iteration and a lot of development and communication over multiple teams is required. On top of the costs for the software maintenance, which is much higher then the costs for the initial development, the costs for model maintenance is added to the existing costs [AA20]. This is where the need for MLOps originates with the goal to automate processes in the ML workflow so "[...] at it's core, MLOps is the standardization and streamlining of machine learning life cycle management" [Tre+20]. Following this goal, DevOps concepts to standardize and automate software development are integrated to ML production frameworks. DevOps aims to reduce repetitive software integration work done by developers. This is reached by automating tests and deployment, introducing postdeployment verification processes to detect and resolve errors and enabling Continuous Deployment (CD) which liberates developers to deploy code automatically to customer related production en-

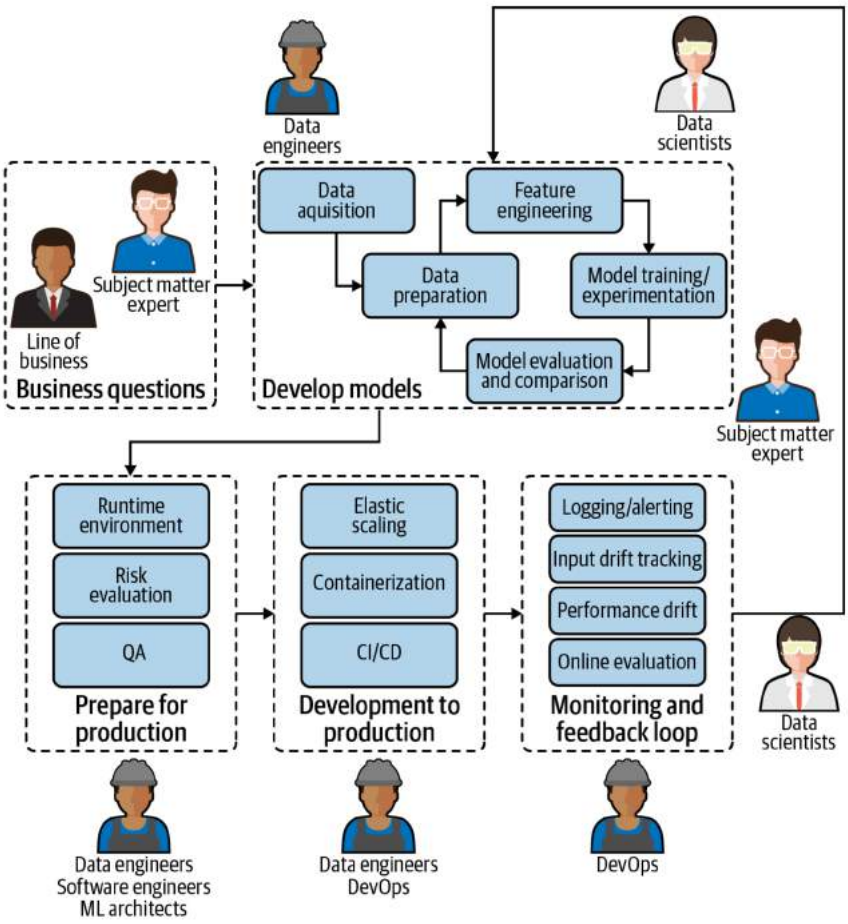


Figure 2.2: Realistic Workflow of Model Production and Maintenance [Tre+20]

vironments without coordinating with other development colleagues. Additionally "DevOps practices rely heavily on tools of various kinds, including tools for container management, continuous integration, orchestration, monitoring, deployment, and testing." [ZBCS16]. Continuous Integration (CI) is a software development practice where code gets published multiple times per day with often minor improvements to reduce the time between releases and improve quality and productivity. The final integration to the production environment often remains manual though after having produced production ready code [SABZ17]. The required deployment cycle of new models fulfilling current business demands can vary with the data collection interval and changing business requirements but the most extreme deployment cycle is the *online learning* [ACBG02] where the production model gets updated with every user request. In practice, batch processing is used more often to allow validation of the new data and new models before being deployed to production [Bay+19].

Since for many organizations, creating and deploying multiple ML models is a relatively new concept, the number of models maintained is limited and therefore manageable without a sophisticated management concept. With automated business decisions that are interacting with multiple parallel running models, managing models and related risk becomes more important and more difficult. These risks can include having a model which is not available for some time, getting bad model predictions or reduced model fairness. Another possible risk is loosing essential personell for maintaining the model [Tre+20]. This leads to a growing need and interest for reliable and efficient production and maintenance of ML applications and their infrastructure in form of MLOps (see fig. 2.3).

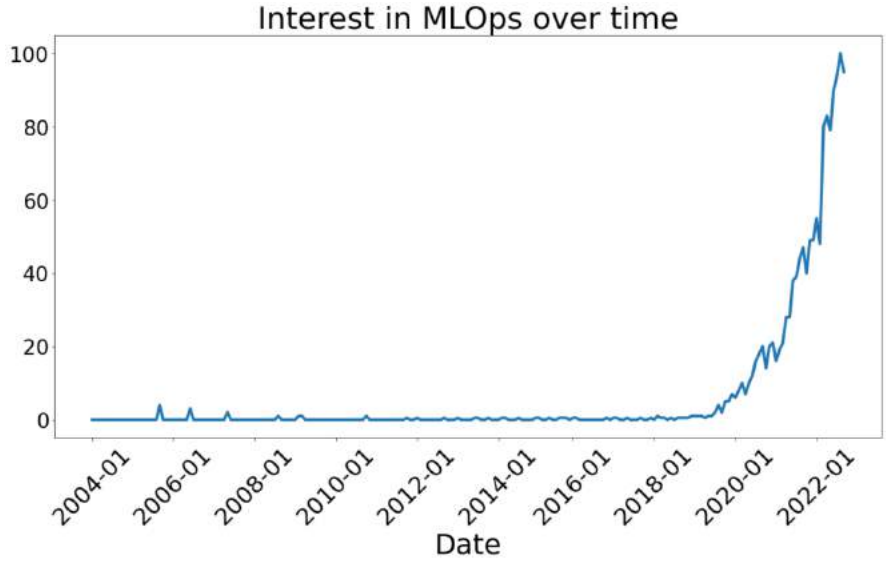


Figure 2.3: Google Trends Worldwide Results for 'MLOps' Search Scaled to Maximum 100, Monthly Results up to 2022-07, Peak in 2022-06

2.1.1 *ML Ops Levels*

MLOps architectures can differ in their level of automation and integration of code and modules within the framework. The most basic level is the **manual implementation** where all steps of the described ML workflow are performed by hand in a sequential manner and models are manually deployed and implemented into the application by data-/software engineers and software architects. Operational teams are responsible for maintaining functionality and collecting monitoring data [AA20]. With periodic required model adjustments or retrainings, this setup will result in repetitive manual adjustments of the model and possibly the engineering steps to integrate the model into the application.

The **continuous model delivery** level introduces pipelines to enable the integration of different parts of the MLOps workflow into a partially automated framework. The goal is to automatically trigger a retraining of the model after a retraining signal has been given and to deploy this model to the production environment. This retraining signal can be given manually or automated via schedule, degrading model performance or changing data patterns. New data is constantly or periodically added to the data store where data preparation pipelines clean and process the data to add fresh data to the feature store where more pipelines draw processed data from [AA20]. This serves the purpose of creating a standardised procedure of data preparation over multiple model versions. The model creation can contain several pipelines prepared to create different model types with various requirements regarding e.g. data format. With these pipelines, models can be built, trained, validated and tested automatically including the tuning of hyperparameters and optimizing performance. At some steps, manual input might be required with more complex models but automation should perform most of the work to generate and deploy an optimal model [AA20]. To achieve this integration and interchangeability of models, the pipeline code needs to be generic so that an expected standardized input is given and the output can be processed by the following pipelines without manual adjustments. This code modularization also enables components to be reuasable and potentially shareable across ML pipelines. Ideally, code is containerized to allow code execution independently from the runtime it was created in [Clo22]. After receiving a retraining signal and having produced an updated model, it gets added to a model store often referred to as *model registry* where storage and versioning of models and weights takes place and the best performing model can be deployed to the application [AA20]. At this level, new pipelines are tested manually before being integrated in the framework and usually only a few pipelines are used to serve the automated retraining and deployment of models with new data available [Clo22].

CI/CD Pipeline automation is the next step in MLOps hierarchy and aims to equip data scientists with the necessary tools to quickly build, test and deploy new pipelines for novel model architectures or feature engineering ideas [Clo22]. This CI and CD of pipeline code improves the prior level in the

extend that no manual testing of new pipeline code and it's components is needed. Changes of state-of-the-art model architectures or data requirements can be adopted rapidly and can serve in combination with CD of the model as an optimal ML environment [AA20].

2.1.2 *Provider and Platforms*

Many data scientists and companies like NVIDIA, Facebook, Spotify or Google [VM21] have a demand for MLOps solutions to be able to constantly adjust their ML models to new data, changing user behaviour or new trends in model architectures. This increasing demand for MLOps opens the possibility for several providers to offer their solutions regarding pre-built MLOps environments. Some of them can be integrated in On Premises (On-Prem) architectures and others are build by or use the resources offered by cloud computing companies.

One entry level open-source tool is *MLFlow* [1] which offers automatic modularization and pipeline creation to experiment collaboratively with different models in an On-Prem environment, build staging zones for models where colleagues can inspect, comment or adjust provided models before they can be easily deployed to the cloud environments of AWS, Azure or Google [AA20]. Monitoring of deployed models is not possible per default so there can be no sophisticated CI/CD for pipelines and models without extensive custom pipeline generation and maintenance. Other On-Prem open source MLOps solutions with comparable functions are SACRED [Gre+17] or DVC [BEA21].

As a more advanced open-source tool, *Kubeflow* [2] is a Google initiated platform to operate with ML workflows on Kubernetes. Containerized ML components can be managed and deployed in a distributed manner to achieve MLOps tasks like training, deploying, monitoring, logging, alerting, retraining and redeploying inside a Kubernetes cluster. Using Kubeflow, the technical overhead of the Kubernetes cluster management should stay at a minimum level and offer a quickly accessible MLOps framework [Bis19]. Kubeflow can be installed in an On-Prem Kubernetes clusters but also in cloud environments using Kubernetes which simplifies the portability between different infrastructures [ZYD20].

If no Kubernetes Cluster is available or desired, Tensorflow Extended [Bay+17] is another open-source MLOps framework by Google offering MLOps functions with the Tensorflow training library but also gives the opportunity to implement other learner algorithms.

There is also a landscape of commercial providers offering MLOps solutions like Valohai [3], Neptune [4], Iguazio [5], Databricks [6] or DataRobot [7], which of-

1 https://mlflow.org/
2 https://www.kubeflow.org/
3 https://valohai.com/
4 https://neptune.ai/
5 https://www.iguazio.com/
6 https://databricks.com/
7 https://www.datarobot.com/

ten provide frameworks leveraging cloud computing resources and providing attractive GUIs. Choosing those providers, additional costs to the sole computing resources pricing are to be expected.

Conducting experiments and projects using ML algorithms and tools, voluminous data storage capacities as well as considerable computing resources in form of CPU, GPU and TPU are required, especially in the training phase [Jou+17; Sze+17]. Subsequently, platforms offering MLOps capabilities need to provide those resources to combine ML and DevOps capabilities [ZYD20]. Since cloud computing companies naturally offer these computing resources and a broad scope of integrated algorithms and services [PIP16], they have an advantage in offering MLOps framework over independent commercial providers. For the customer it can be also beneficial to rely on cloud computing resources because of the availability of large computing and storing volumes on demand coupled with the costs proportional to the actual usage as well as scalability and reliability [PIP16]. A lot of risks for MLOps practitioners can be eliminated by evading the costs of an On-Prem computing environment including costs of hardware, setup, maintenance and backup management. Additionally a lot of state-of-the-art resources and knowledge becomes available very easily which especially in countries or regions without the availability of those resources can make a big difference for customers. These resources can include available hardware for the required computing or storage capabilities as well as highly skilled personell for development and administration. On top of that, time needed for acquiring, building and maintaining the infrastructure can drastically be cut down so more time becomes available to focus on building the operations planned on that environment. The benefits of cloud environments can be listed as [Sun20]

- Low Entry Barriers
- Pay-as-you-go
- Access to Leading Edge IT Resources, Skills, and Capabilities
- Quality Improvements
- Cost Savings
- Focus on Core Capabilities
- Greater Flexibility and Elasticity
- Reduced Time to Market and
- Lower IT Barriers to Innovation.

As downsides, lower costs in this case also mean sharing resources, giving (security) control out of hand, external availability management, additional required platform knowledge, vendor lock-in and limited service continuity [Sun20]. For a lot of companies, this trade-off makes experimenting with possible ML solutions much easier and possibly more affordable when using on

demand cloud solutions. This results in a sales volume of about 411 billion US$ in 2021 regarding public cloud computing services [Sta22].

In an analysis of the biggest competitors in Cloud Infrastructure and Platform Services, Bala et al. [Bal+21] build a Magic Quadrant visualization (fig. 2.4) to cluster these companies regarding their momentary roles in the market differentiating between Challengers, Niche Players, Visionaries and Leaders. To be present in the magic quadrant, companies must fulfill several requirements such as being among the top global providers for segments like industrial and private cloud infrastructure and platform services. Companies older then 3 years are required a minimum revenue of US$1 billion in the previous year or at least US$500 million when younger then 3 years. The axes completeness of vision and ability to execute are each a weighted average of multiple evaluation criteria. Completeness of vision consists of: market understanding, marketing strategy, sales strategy, product strategy, business model, vertical/industry strategy, innovation and geographic strategy. The ability to execute is evaluated by the criteria: product or service, overall viability, sales execution/pricing, market execution, customer experience and operations. In their 2021 evaluation, there are 3 leaders identified: AWS, Microsoft (Azure) and Google (Google Cloud Platform). Those market leaders in Cloud Infrastructure and Platform Services all offer a broad scope of resources and services including MLOps capabilities. Because of the dominating relevance of AWS as the market leader, an overview of how an MLOps project can be implemented with the services offered by AWS is given in chapter 3.1. This overview is followed by the discussion of a possible weakness in that framework when working with time series data and the introduction of a possible solution to that problem.

2.2 TIME SERIES FORECASTING AND MONITORING

Forecasting data or events is a highly demanded operation in various fields like politics, industry, medicine, science or finance. Due to Montgomery, Jennings, and Kulahci, most of the forecasting problems include time series data which they define as "[...] a time-oriented or chronological sequence of observations on a variable of interest" [MJK15]. The temporal distance between the observations of one time series is fixed but the spacing between those observations can vary between fragments of a second, possibly with industrial sensors, to many years, like the evaluation of CO_2 levels over time in drilled ice cores [Fou22]. Predictions based on a time series or multiple time series of a similar type can vary in their *prediction/forecast horizon* as well, meaning the industrial sensor with an observation every second could get a forecast just for the next second or for the next e.g. 1000 seconds. So forecasting problems can be categorized into short-term forecasting problems with just one or a few time periods into the future, medium-term forecasting problems extend from 1 to 2 years into the future and long-term forecasting problems can largely extend that period [MJK15]. One data point in a time series can embody a snapshot of an actual event like the momentary rounds per

Figure 2.4: Gartner [Bal+21] Magic Quadrant for Cloud Infrastructure and Platform Services

minute of a spinning machine or the cumulative amount of revolutions this machine made during the day. Generating predictions with time series data can help making short-term or strategic decisions in different scenarios. In quantitative forecasting, historical data and their inherent patterns are used to build a forecasting model with statistical means and to extrapolate the observed data into the future. A prediction can then have the form of a *point forecast*, so an exact value which will produce a *forecast error* almost certainly or provide a *prediction interval* to accompany the point forecast [MJK15].

There are many ways, forecasting models can be built depending on the forecasting problem, the available data and the resources at hand for building and maintaining that model. To evaluate the performance of a model, one could calculate the forecasting errors of the data the model was built on. This *in-sample* error only describes the model goodness of fit but what is more important is the accuracy of the future predictions, the *out-of-sample* forecast error [MJK15]. This error should be evaluated initially to decide if a model performs according to the goals or expectations of the business or research problem. If the model does not perform in a satisfying way, adjustments or even a completely different model type are required. After having decided on a model and having deployed it, the evaluation of the model performance must continue over time to assure the accuracy is not degrading over time. This is done by *monitoring* the model. There are multiple metrics to assess the accuracy of a model but commonly a single forecast error is calculated as *one-step-ahead* forecast error at timestep t

$$e_t(1) = y_t - \hat{y}_t(t-1), \tag{2.1}$$

where $\hat{y}_t(t-1)$ is the forecast of y_t that was made one period prior. $e_t(1)$ is the one-step-ahead forecast error at timestep t. This is done because changes in the time series like a level change will typically be also visible in the forecast error [MJK15]. Using larger forecasting horizons mostly leads to a more difficult problem due to the increasing uncertainty [Wei18].

2.2.1 *Time Series Data*

Due to the definition of time series observations coming from one entity, the observations are correlated and therefore standard statistical methods requiring random samples (independent and identically distributed random variables) are not applicable [Wei13]. Another characteristic of time series data is the question of their *stationarity*. A time series Z_t is *strictly stationary* when the joint distributions of the observations $x_1, ..., x_n$ is the same as the joint distributions of the observations plus any integer k to generate a time shift, so the distribution of a time series is invariant to a change of time origin:

$$F_{Zt1,...,Ztn}(x_1, ..., x_n) = F_{Zt1+k,...,Ztn+k}(x_1, ..., x_n) \tag{2.2}$$

for any tuple $(t_1, ..., t_n)$ where $F_{Zt1,...,Ztn}(x_1, ..., x_n)$ is the joint distribution function defined by $F_{Zt1,...,Ztn}(x_1, ..., x_n) = P\{Z_{t1} \leq x_1, ..., Z_{tn} \leq x_n\}$. Because of the impracticability to check that on a complete time series, often a *weakly stationary* process, which only requires a time invariant mean and variance is considered sufficient to classify a time series as stationary. To identify stationarity, visual inspections when plotting the data or statistical tests can be used [Wei13].

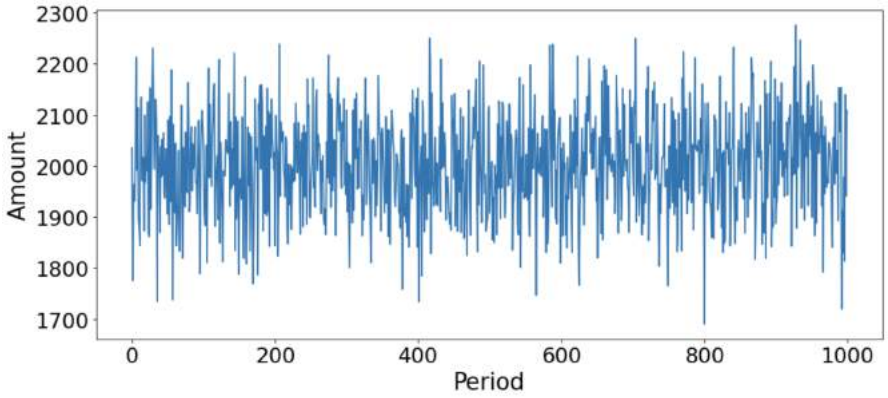


Figure 2.5: Stationary Time Series

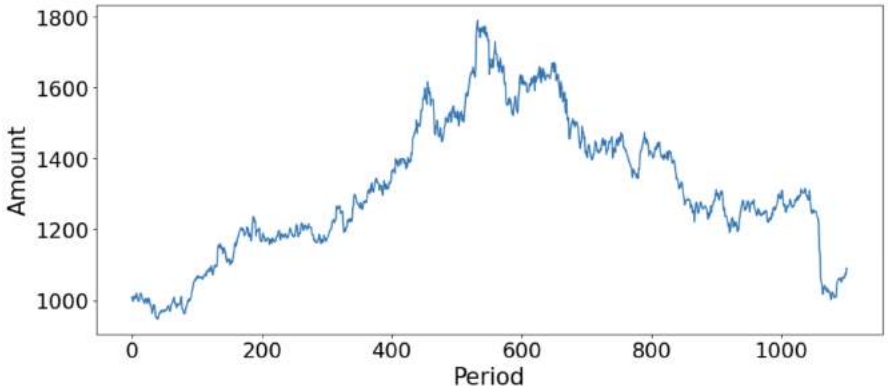


Figure 2.6: Non-Stationary Time Series

Two more important concepts when modeling time series data are *seasonality* and *trend*. The time series in (fig. 2.7) clearly displays an upward trend over time and simultaneously a seasonal pattern of peaks and valleys is visible with a span of about 20 periods.

2.2.2 *Time Series Forecasting Models*

There are several types of models that can be built to serve as a forecasting instance for time series data. Generally those can be divided into statistical models using more classical methods and ML models which are becoming popular more recently. Hybrid models aiming to combine the advantages

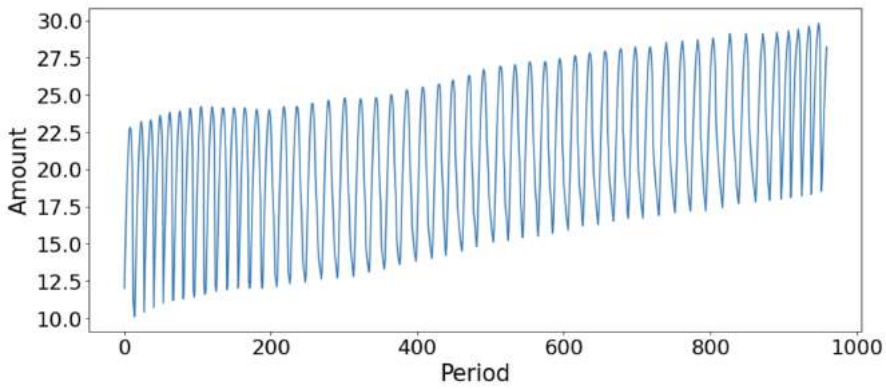


Figure 2.7: Example Time Series for Seasonality and Trend

from both worlds are being developed as well, some of them challenging the state-of-the-art performance [MSA18b].

2.2.2.1 *Statistical Models*

The most basic type of statistical time series forecasting model is the *naïve* method also called random walk forecast which predicts the future solely on the last known observation producing a practicable fit for financial data [KT03] (source notation altered for uniform naming convention):

$$\hat{y}_{t+1} = y_t \tag{2.3}$$

Some of the most popular time series forecasting models used today rely on statistical foundations layed out in the 1950s and 1960s by Brown [Bro59; Bro63], Holt [Hol57] and Winters [Win60]. These *exponential smoothing* approaches can partly model time series data with trend and seasonality where "Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation the higher the associated weight." [Ath14].

There are different types of exponential smoothing models available with the *simple exponential smoothing* as the most basic variant for univariate data with no trend or seasonality. It only requires one parameter *alpha* often referred to as *smoothing factor* or *smoothing coefficient* that influences the exponential decay of the weights of older observations where "A value close to 1 indicates fast learning (that is, only the most recent values influence the forecasts), whereas a value close to 0 indicates slow learning (past observations have a large influence on forecasts)" [SJ16].

The two extensions of this method are *double exponential smoothing* and *triple exponential smoothing* where additional parameters in the model enable the modeling of trends and seasonality. The double exponential smoothing model with an additive trend is often referred to as *Holt's linear trend model* [Ath14], but it is also possible to model exponential trends. The triple ex-

ponential smoothing model is also called *Holt-Winters Exponential Smoothing* and the most sophisticated type of exponential smoothing being able to model linear and exponential seasonalities. To improve modeling of the seasonality, the span of the seasonal pattern needs to be provided when building this model [SJ16].

Exponential smoothing models have widely been used since then and experienced some improvements and further variants especially in the 1980s leading to a remarkably good forecasting performance [DH06].

One of these variants is the ETS model type which can additionally to a point forecast give predictions for intervals. Every ETS model consists of an equation that describes the observed data as well underlying unobserved components like level, trend and seasonality as well as the change of those components over time. Because of that emphasize of changing components, these models are also referred to as (innovation) *state space models* with the name-giving components *error*, *trend* and *seasonality* [HA18]. To build an ETS model, maximum likelihood estimation is used in combination with obtaining a good Akaike's Information Criterion (AIC), Bayes Information Criterion (BIC) or mostly Akaike's Information Criterion corrected value [JM17] which are all used to assess the quality of a model while considering simpler models as beneficial.

Another established type of statistical time series forecasting modeling is the ARIMA model group. Box and Jenkins [BJ70] combined the existing concepts of Autoregressive (AR) and Moving Average (MA) models first introduced by Yule [Yul27] to develop their "[...] three-stage iterative cycle for time series identification, estimation, and verification (rightly known as the Box-Jenkins approach)." [DH06].

These ARIMA models require 3 parameters to combine the subfunctions of the model into a whole entity: the parameters p, d and q describe the order of the AR, I (Integration) and MA parts. The AR part describes a linear combination of the past values of the time series at a desired point. In other words a regression on the variable itself is build. Just like any other linear regression models, it is best if the predictors are independent and uncorrelated. This is why the I part of the ARIMA model describes the degree of differencing required to turn a non-stationary time series into a stationary time series without any trends or seasonality. The MA, has a comparable behaviour like the AR part in that sense that they both are a linear combination of past values, but instead of using the predictor itself, the MA relies on the most recent forecasting errors building a weighted moving average. Using these parameters, non-seasonal time series data can be modeled but with additional parameters, also seasonal data can be portrayed [HA18].

To find the best parameters to build an ARIMA model, the dependency on the architects knowledge and experience can be eliminated by an automatic search process comparing different combinations of those parameters and the results regarding their one-step-ahead prediction errors while penalizing overfitting. For this model comparison popular metrics are the AIC, FPE (Akaike's final prediction error) or the BIC [DH06]. There are multiple vari-

ants of ARIMA models available like *ARARMA*, multi-step-ahead forecasting AR models selected separately for each horizon, multivariate *VARIMA* models, long memory *ARFIMA* models or different model types like ARCH/-GARCH models focusing on the variance of time series data [DH06], which will not be further discussed here.

2.2.2.2 *ML Models*

In the 1990s a different approach to time series forecasting joined the state-of-the-art discussion: ML models, often relying on artificial neural networks. This architecture was supposed to bring a novel approach to the forecasting, especially of nonlinear processes with an unknown functional relationship [DH06]. The inputs for the input layer are values of the observed time series, processed with the learned weights and biases in one or more hidden layers to give one feed forward output as a prediction for the next data point at the desired interval. There exist different types of ML time series forecasting models but several studies observe an inferior predictive performance when being compared to statistical methods [MJK15; DH06; MSA18b; MSA18a]. ML models used in those comparisons are multiple types of neural networks (e.g. Long Short-Term Memory (LSTM), multi-layer perceptron), decision trees, support vector regressions, nearest neighbor methods and Gaussian processes. Cerqueira, Torgo, and Soares [CTS19] find that this prevailing opinion of inferior performance exists due to the low sample sizes used in comparative studies. Focusing on the small training sets used in the M3 forecasting competition [MH00], they show that with larger training sets, several ML forecasting models outperform statistical models when using univariate time series. The M3 competition is the third iteration of the M competition series initiated by Spyros Makridakis to evaluate state-of-the-art models and algorithms in time series forecasting. The M3 and the M4 competition datasets have a comparable structure providing multiple univariate datasets with e.g. yeary, monthly and daily intervals. The M3 data includes 3003 time series of limited length (e.g. all 1428 monthly time series below 150 observations - see fig. 2.8) and aims to include more methods, especially neuronal networks [MH00]. The main differences to the M4 datasets is that the M4 dataset is with 100.000 time series much bigger, the time series are much longer and shorter intervals (daily & hourly) are provided additionally (see appendix chapter A.1). One of the main objectives is to further evaluate which models, statistical and ML models, perform well in which scenario and when combinations of these methods perform best [MSA18b]. Since the M4 competition [MSA18b] has larger training sets but still pure ML models show a worse predictive performance then statistical models, a validation of the results found by Cerqueira, Torgo, and Soares [CTS19] is required. The strength of ML models at the state-of-the-art research can be shown either with combining ML and statistical models [MSA18b] or using ML models as global models (cross-learning) so to learn not only from one time series but from many provided these time series are correlated [MSA22; HBB22]. The most recent M competition, the M5 competition, even finds that a combina-

tion of pure ML approaches can lead to superior forecasting accuracy using models like LightGBM, DeepAR or N-BEATS [MSA22]. The data used in the M5 competition is sales data provided by Walmart regarding product prices (see fig. 2.9). The data has a hierarchical order and additional information to the time series is provided e.g. about the day of the week, inventory levels or promotional activities [MSA22]. These findings show that using ML models in time series forecasting continues to act as a promising approach to improve results.

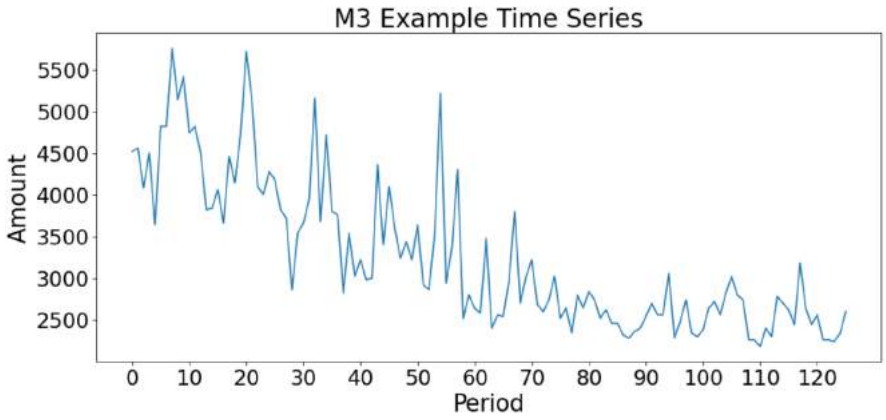


Figure 2.8: Example Time Series from M3 Monthly Dataset - Index N1724

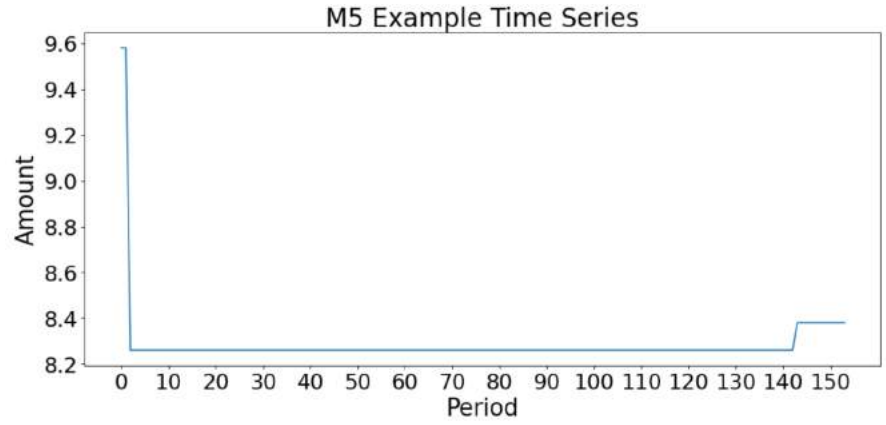


Figure 2.9: Example Time Series from M5 Dataset - Item ID HOBBIES_1_001 at Store CA_1, The Dataset Shows Less Volatile Time Series then the M3 and M4 Datasets

2.2.3 *Monitoring Metrics*

Evaluating the performance of a time series forecasting model is a crucial part of the forecasting process. When building a model, it must be assured that the predictive performance of the selected model exceeds that of a random or naïve process and that the selected model has the best performance of all considered candidates. When having deployed a model, a continuous quality control should be established to assure the predictive performance

stays at the desired level and does not decrease beyond a defined limit. For this performance evaluation and monitoring, appropriate metrics can help within a data driven decision making process. When evaluating model performance at the initial model selection, it is important to take into account that the evaluation of the model should not happen on the training data, also called *in-sample* prediction or goodness-of-fit approach because the quality of a model fitting it's training data "[...] does not really reflect the capability of the forecasting technique to successfully predict future observations." [MJK15]. Instead, a comparison of the *out-of-sample* forecast errors leads to a model with promising forecasting capabilities.

When evaluating n out-of-sample observations for which one-step-ahead predictions have been made, the forecast errors would be $e_t(1)$ with the standard forecasting accuracy measures [MJK15]:

average error or *mean error*

$$\mathrm{ME} = \frac{1}{n}\sum_{t=1}^{n} e_t(1), \tag{2.4}$$

mean absolute deviation or *mean absolute error* (MAE)

$$\mathrm{MAE} = \frac{1}{n}\sum_{t=1}^{n} |e_t(1)| \tag{2.5}$$

and *mean squared error* (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{t=1}^{n} [e_t(1)]^2 \tag{2.6}$$

where the ME as expected forecast error should stay close to 0 when seeking for an unbiased forecast. When a drift away from 0 is noticed, a change in the regarding time series happened which the model could not process as good as before. The MAE and the MSE both describe the variability of the forecast error which should stay as small as possible [MJK15]. Since those metrics all are scale variant and can only be interpreted when knowing the unit of the time series, scale invariant metrics are required to offer a comparison of predictive performance between time series of different scales. Based on the *relative forecast error* or *percent forecast error* [%]

$$\mathrm{re}_t(1) = \frac{y_t - \hat{y}_t(t-1)}{y_t} 100 = \frac{e_t(1)}{y_t} 100 \tag{2.7}$$

there are two established scale invariant measures, the *mean percent forecast error*

$$\text{MPE} = \frac{1}{n}\sum_{t=1}^{n} \text{re}_t(1) \tag{2.8}$$

and the *Mean Absolute Percent Forecast Error (MAPE)*

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n} |\text{re}_t(1)| \tag{2.9}$$

The selection of an appropriate monitoring metric is a task often underestimated due to the specific requirements of the mentioned and many other available metrics [DF13]. Davydenko and Fildes give a thorough overview about the weaknesses and areas of application of specific error measures to monitor a time series forecasting process. Concerning percentage errors like the MAPE, they emphasize possible weaknesses when the actual value in the denominator y_t is small or even zero which results in the MAPE becoming very large or even undefined. There can also be a bias to overrating the performance of a model so as general requirements to an error measure, they define scale-independence, robustness to outliers and interpretability, admitting that scale independence measures are not always desirable.

When having found an appropriate error measure, it can then be used to monitor the performance of a forecasting system and produce alerts when some kind of outbreak is detected. This outbreak can be understood as a trend or level change in the underlying data for example a disease outbreak. So to give a helpful warning in case of an outbreak, the alert "[...] must occur quickly after the outbreak begins, should detect most outbreaks, and should have a low false alert rate." [LS09]. To produce an alert, often forecasting errors are in some kind applied to a control chart where upper and lower control limits are introduced to function as alerting threshold. The most used types of control charts are the Shewhart control chart, the Exponentially Weighted Moving Average and the Cumulative Sum [LS09] explained in detail by Montgomery [Mon07].

Another approach to monitor a time series forecasting model is by using a tracking signal. In general, tracking signals are defined as a ratio with the nominator as a sum or weighted sum of forecast errors that should stay close to zero in a well working forecasting model and the denominator with the purpose to normalize by the long-term average variability of forecast errors [GM05]. With this quantitative measure, an automatic monitoring of forecast errors can be established that gives a retraining signal based on a given threshold when an existing pattern or relationship has changed [DJI04]. Advancing on the examination by Brown [Bro62], who computes a tracking signal by dividing the sum of the forecast errors by the MAE (see equation 2.5), Trigg [Tri64] introduces a tracking signal (TS) that comes back in control after an outbreak without needing to reset the cumulative error sum. He

uses a smoothing parameter α to adjust the long term memory of the MAE in the denominator. This solves the problem of manual interference with the cumulative error sum and potential infinite values of the tracking signal in case of a recently nearly perfect forecasting resulting in an infinitely small denominator:

$$\mathsf{TS} = \frac{\mathsf{Errorsum}}{\mathsf{MAE}} \tag{2.10}$$

where Errorsum = previous sum of errors + latest error (smoothed)
and MAE = (1 - α) * previous MAE + α * latest absolute error (source notation altered for uniform naming convention). This equation is a schematic representation to introduce the tracking signal more clearly defined in equation 2.11.

McClain [McC88] found that this tracking signal is a good choice for practitioners but improved Trigg's tracking signal ratio in the denominator to further emphasize the purpose of measuring the long-run average variability of forecast errors in order to standardize the numerator. He used a separate smoothing parameter in his research, which is used in more modern monitoring systems [DJI04; GM05] and is also examined in the experiment conducted in this thesis:

$$TS = |\frac{\alpha_1 e_t + (1-\alpha_1)E_{t-1})}{\alpha_2|e_t| + (1-\alpha_2)MAE_{t-1}}| \tag{2.11}$$

where e_t is the forecast error, the nominator is E_t as smoothed errorsum and the denominator is the smoothed MAE_t. Initial values set for E_t and MAE_t can result in a burn-in phase until reasonable results are computed (source notation altered for uniform naming convention). During the burn-in phase the initially set values for E_t and MAE_t, which are only an approximate guess, change in an iterative process by calculating these parameters with actual observations getting closer to the real value in each step. Investigations like by Cohen, Garman, and Gorr [CGG09] evaluate fitting values for α_1 and α_2 in an experimental setup.

2.3 DRIFT

The decrease of the predictive model performance over time can originate in different sources and result in negative business effects. A requirement for good performance monitoring is a correct ground truth for every forecast being available in an acceptable timely spacing to the forecast. Another attempt to monitor the performance is by only observing the input coming to the model as request. If the training data differs strongly from the recent data coming to the model, a deterioration of the performance can be expected [Tre+20]. Since this thesis is comparing a time series forecasting model monitoring with tracking signals in an MLOps environment to the

state-of-the-art technology, which is using AWS as a platform for this endeavour, the monitoring possibilities offered by AWS will be described. Those include the performance monitoring approaches introduced earlier but extend those as well.

AWS offers 4 types of monitoring for models produced with AWS SageMaker, which is a fully managed machine learning service [Ser22n]. These 4 types can continuously monitor the model performance and produce alerts that can then lead to a model retraining and redeployment: monitor data quality, monitor model quality, monitor bias drift for models in production and monitor feature attribution drift for models in production [Ser22l].

2.3.1 *Data Drift*

Monitoring data drift is a simple way to monitor a forecasting model that can very quickly react to possible performance decreases since no new ground truth labels are required. When training the model, a baseline dataset with statistics of the training data is generated. These statistics include datatypes, distributions and other descriptive statistics like mean, sum, standard deviation, min and max. With these statistics, a constraint file is generated where these metrics and completeness of columns are incorporated. When new data is send to the deployed model to produce forecasts, the requests are logged and periodically analyzed. If in this batch process violations of the constraints like different data types, disproportionately missing values or a changed value distribution within a column are recognized, an alarm can be triggered [Ser22j]. Since the actual influence of the change in predictive performance is unknown when monitoring data drift, additional monitoring approaches should be implemented. When short term reactions to changing data are necessary or no updated ground truth data is available after deploying a model, monitoring data drift becomes essential.

2.3.2 *Model Drift*

Monitoring the model quality directly means "[...] comparing the predictions that the model makes with the actual ground truth labels that the model attempts to predict." [Ser22k]. To achieve that in a deployed model, the model requests need to be stored and then merged with the ground truth when supplied to the monitoring system at a later point. The timely gap between generating a prediction and comparing the result to the ground truth can vary greatly between seconds to many years depending on the desired forecasting task. As a more common estimation of the frequency of new ground truth and by that also new training data becoming available, a daily basis or less seems reasonable [Tre+20]. The process of gathering ground truth data can be challenging but in the case of time series data, this should be no major problem since one must only collect more recent data of the same time series the forecasts are made for.

When forecasts and ground truth data are merged, one or multiple metrics need to be defined in order to calculate the error of the model. When producing forecast for time series, error measures for regression problems can be applied. A classification problem could also be established when the model is tasked with alerting only when a step change in the data is detected but in most applications a metric forecast of the time series is desired. AWS provides 4 error metrics when comparing forecasts and ground truth in a regression problem: MAE, MSE, RMSE and R^2. Additionally a standard deviation is calculated each by randomly sampling 80% of the data 5 times [Ser22g]. These metrics are calculated in a batch process for the most recent data points where ground truth labels are available for and are then compared to the same metrics calculated on the baseline dataset, where "Typically, you use a training dataset as the baseline dataset" [Ser22f]. If a performance decrease is detected at any point, an alert gets produced which can then trigger a retraining of the model with more recent data.

2.3.3 *Bias Drift for Models in Production*

The Bias Drift for Models in Production is another method of alerting for possible forecast performance deterioration without analyzing new ground truth data. With the training set, a baseline is generated that gives statistics and threshold suggestions. Within these limitations, a range is defined that is used to control if forecasts produced by data used in inference of the deployed model are valid. If the produced forecast is not in range of the suggested boundaries, an alert is produced [Ser22h]. This kind of drift as well as the following drift will not be further discussed but explained briefly to complete the overview of possible drift monitoring methods.

2.3.4 *Feature Attribution Drift for Models in Production*

With the Feature Attribution Drift it becomes possible to monitor if the realisations of a feature have prominently changed compared to the realisations in the training data [Ser22i]. Since this approach only relies on the drift of the input data, no new ground truth data nor the results of model inference are required. So if e.g. the highest educational qualification feature in the training data is mainly high school level but later on in deployment most of the requests sent to the forecasting model show an academic level in this feature, an alert can be raised. This kind of monitoring seems unfit to be used with univariate time series forecasting models.

Since this approach only relies on the drift of the input data, no new ground truth data nor the results of model inference are required.

3

RESEARCH HYPOTHESIS

When large ML projects with multiple teams or at least multiple team members are executed, a framework to organize the different parts of the project and the reuse of as many framework parts as possible are requirements to succeed in fast changing environments or a constantly changing business demand. Automation should also be implemented as much as possible to free up development time for data scientists, data engineers, DevOps specialists and other team members. This is why MLOps is experiencing the extreme increase in demand shown and explained in chapter 2.1. Univariate time series forecasting is a problem field where monitoring the predictive performance is an essential part of the forecasting process since the underlying data trends can quickly change and result in the need for an adapted forecasting model (see chapter 2.2). Due to the relatively new existence of MLOps architectures, the implementation of the monitoring of such time series forecasting models in an MLOps environment is a young research topic. The utilization of cloud computing resources brings many advantages where easy accessability and nearly endless scalability options are benefits attractive to many researchers and companies. Since it has been indicated that AWS is the leading provider of cloud computing resources and services (fig. 2.4), the way how MLOps coupled with time series forecast monitoring can be implemented in an AWS environment is an important benchmark. This thesis provides an overview about how this MLOps framework can look when implemented at AWS and examines a possible weakness in how the model quality monitoring is implemented per default (see chapter 3.2). As an alternative, the implementation of a tracking signal for model quality monitoring is benchmarked against the default AWS solution. Hereby, not only the predictive performance of different models is observed but a novel approach of comparing estimated costs regarding the computing time on the cloud resources and the ecological costs of the resources usage is introduced. This aims to evaluate beneficial timings of when to retrain a time series forecasting model regarding the resulting costs. Implementing pipelines to retrain deployed models can be described as a standard MLOps usecase [Tre+20]. However, the state-of-the-art research is not picking up the question of when hyperparameters need to be adjusted in a time series forecast task but rather hyperparameter-fixed models are compared with their one-step-ahead prediction accuracy ignoring the resulting economical and ecological costs (e.g. [MSA18b; MSA22]) which gives another reason for the research conducted here.

Our **hypothesis** is that the usage of the tracking signal will result in a better cost-benefit relation since it is expected that the amount of retrainings triggered with the AWS default approach will be very high and at almost

every data point. This is expected because the performance thresholds are set as error metrics from the in-sample predictive performance and with unknown data we expect that this performance can rarely be reached. With high economical costs resulting from a decreased model performance, retraining at every data point should be the cheapest solution although the costs for the retraining will be relatively high. Also, it is expected that the tracking signal is a better fit inside an MLOps environment because the customization of the error measure's sensibility is more generic which can further increase the progress towards automation and reusage.

3.1 ML OPS AT AWS

The usage of AWS as an MLOps environment requires several services interacting with one another. There is not one MLOps framework or solution provided at AWS but depending on the individual requirements, services and components can be connected to build the desired framework (compare fig. 3.1). In this chapter, a description of an example framework is given to provide a better understanding of how MLOps can work in a real life scenario.

The surrounding template that defines the different services, resources and interactions is called an AWS CloudFormation [Ser22a] which can be published at the AWS Service Catalog [Ser22c] **(1)** to make that template accessible to other AWS users. Inside of that, there are two pipelines defined with different responsibilities: building the model and deploying the model to production. As heart of the operation, AWS SageMaker [Ser22n] is used to execute the ML code for model training and producing baselines for the monitoring statistics. With the integration of AWS CodeCommit and CodeBuild, a Git-based repository is provided to store, version and build all the pipeline code in a collaborative environment [Ver20] **(2)**. When the pipeline is executed, data from the data storage S3 [Ser22e] is preprocessed, a model is build and baseline statistics are generated. To compare different models and save multiple model versions and artifacts, the produced model gets integrated in the SageMaker Model Registry [Ser22m] **(3)** where new models are placed in a staging area and from there they can be reviewed and approved in a manual or automated manner **(4)**. When a model gets approved, the deployment pipeline is activated and the model is pushed to an endpoint **(5)** where inference requests are processed and saved to S3. The monitoring of the model **(6)** can include up to 4 of the described (see chapter 2.3) monitoring approaches. Only when using the model drift monitoring, the merging of new ground truth data to the monitoring job is required. Otherwise the saved inference requests can be compared to the statistics of the baseline data coming from the training dataset to generate alerts for a potential forecasting performance decrease. This can happen in an AWS CloudWatch [Ser22d] event **(7)** that can result in a CloudWatch alarm that triggers another execution of the pipelines for model building and deployment **(8)**. The execution of a monitoring job can be scheduled with the AWS CloudWatch Schedule Trigger **(9)**.

Another popular service for MLOps operations on AWS not used in this example is AWS Lambda [Ser22b], which is a serverless code execution service that can be used as a trigger based connection between different services.

3.2 OPTIMIZATION OF MODEL DRIFT MONITORING VIA TRACKING SIGNAL

As described in chapter 2.3.2, the model drift monitoring at AWS is done by calculating MAE, MSE, RMSE and R^2 on the training dataset of a model in production. This process generates the baseline statistics for the performance monitoring of the model in production. The deployed model processes inference requests and produces forecasts based on the given data. These requests are stored and when the ground truth data for those requests is available, the request with it's forecast and the ground truth is merged. With a time series forecasting problem, it is often just a matter of time until ground truth data becomes available since the forecasts are just extrapolations of the data used for the inference request. No manual labeling is required. The threshold values from the baseline calculation can be altered so for example a higher MSE could be set as a threshold but per default, the values calculated in the baseline process are used. We criticize this default process because we expect that the error threshold will be exceeded at nearly every new data point because a model that was fit on known data probably does not perform as good on unknown data. Exceeding this threshold results in an alarm that in an MLOps environment triggers a retraining pipeline and potentially a deployment pipeline for the new model resulting in economical and ecological costs that possibly could be avoided when using a different kind of metric to generate alarms for a decreasing predictive performance. Also, the possibility of changing the default value is considered an inappropriate approach for an MLOps environment because when only provided the different error measures it is difficult to foresee the impact on the rate of retraining when adjusting those values. With the usage of a tracking signal we expect a more reliable retraining cycle when a specific threshold is provided so for example when due to relatively high training costs, a lower frequency of retrainings is desired, a tracking signal threshold of 1.8 will provide this approximate relationship independent of the time series data or its scale. This scale invariance leads to a more standardized monitoring approach that benefits the reusability of the monitoring code.

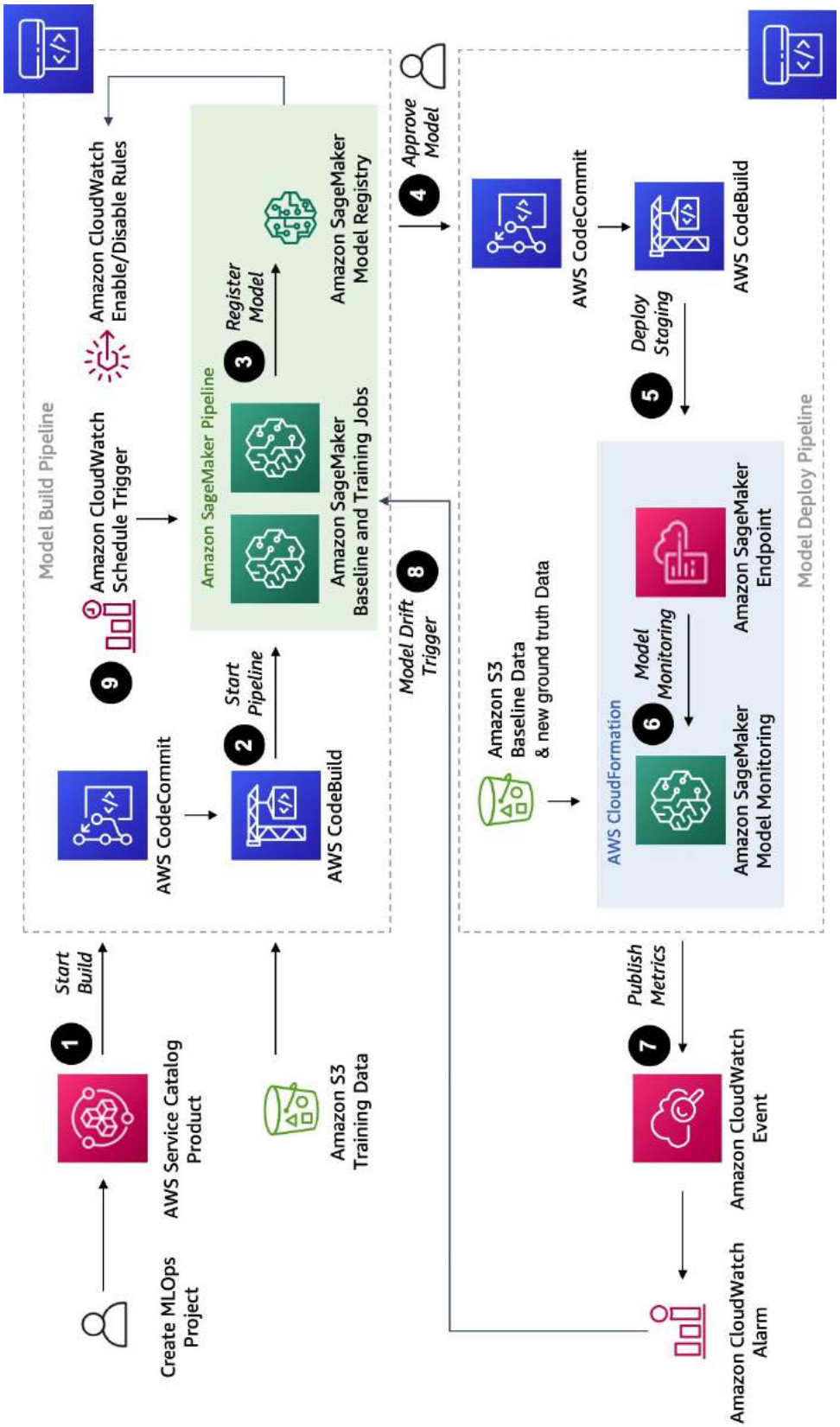


Figure 3.1: AWS MLOps Architecture, Figure Derived from [Bri+21]

4

RELATED WORK

The extrapolation of time series data is a research topic which has been discussed in modern research for at least 70 years and is still highly active today [Bro59; MSA22]. The state-of-the-art research focuses on what kind of models produce the best accuracy, often with one-step-ahead forecasts [MSA18b; MSA22]. There is also active research regarding the challenges and benefits of MLOps environments [ZYD20; BEA21; Bay+19; VM21; Tre+20]. Approaches are discussed about how to reduce economical and ecological costs in cloud computing environments [Yah+21; Nay+18; COH14] but these focus on algorithms that optimize load balancing, workflow scheduling, ressource allocation, security or energy efficient task distribution. The research approach in the intersection of cloud computing, time series forecast monitoring and cost reduction has no comparable existing research. We want to give a brief overview about relatable research concerning tracking signals used for time series forecast monitoring [DJI04; GM05] and two alternative approaches to monitor and select time series forecasting models in an evaluation framework [Can+21; CGG09]. The presented related research regarding tracking signals is also serving as a guideline regarding possible implementations within an MLOps environment.

4.1 SELECTED TRACKING SIGNAL RESEARCH

Deng, Jaraiedi, and Iskander [DJI04] criticize that the research of monitoring intelligent time series forecasting models is not widely and sufficiently addressed. To respond to that problem, they conduct an experiment with a tracking signal as monitoring metric. As univariate time series forecasting model they use a neural network in the form of an Adaptive Neural Fuzzy Inference System (ANFIS) model [Jan93]. After the model is trained, they produce one-step-ahead forecasts by providing the three prior data points x_1, x_2 and x_3 to extrapolate to x_4. To produce three-step-forward predictions the model takes x_1, x_4 and x_7 to output x_{10}.

The dataset they use consists of 100 observations where the first 50 are used for training and the remaining 50 serve as test data. They generate three-step-ahead predictions which are used as inputs for further forecasts until a decrease in model performance is noticed. As a monitoring metric they use the tracking signal introduced in equation 2.11 which is also used in the experiment we conduct.

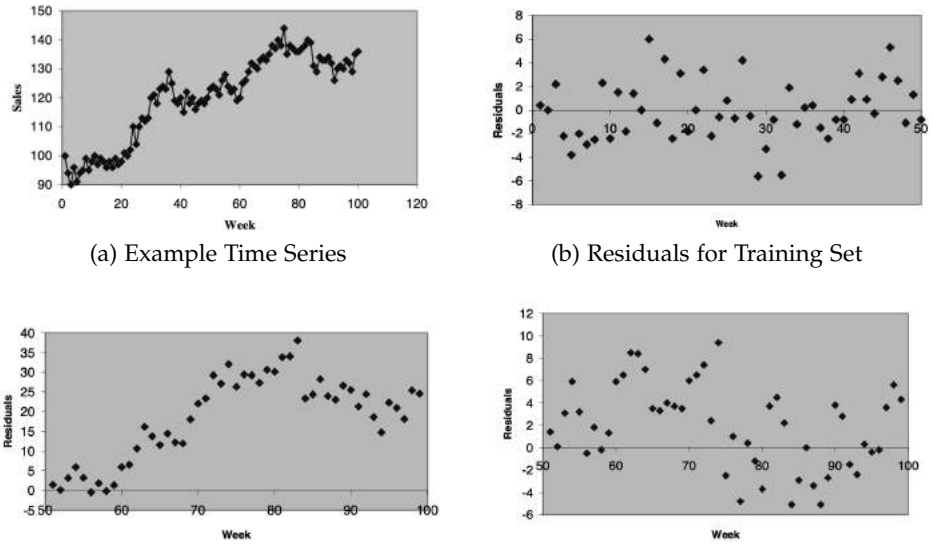
As parameters for the tracking signal they set

- the initial value for $E_t = 0$

- the initial value for M_t is set equal to the average absolute deviation of the training samples
- $\alpha_1 = 0.4$
- $\alpha_2 = 0.2$
- the threshold for the tracking signal to alert at 0.4

When the tracking signal detects that the forecasts are out of control, the new observations available to this point are added to the training data and the amount of those observations are dropped from the oldest observations. After the model is retrained, new forecasts are produced until another cycle of out of control tracking signal, adding of new training data, retraining the model and generating new forecasts happens.

Deng, Jaraiedi, and Iskander find that in their setup and example time series (see fig. 4.1), without the monitoring framework, "[...] forecast errors are increasing over time after a certain number of periods into the forecasting procedure and display a systematic tendency of being positive." [DJI04]. With the monitoring framework, the residuals of the test set display a random pattern which indicates a good model performance also showing a significant MSE decrease from 450.9 to 17.83.



(a) Example Time Series

(b) Residuals for Training Set

(c) Residuals for Test Set without Monitoring

(d) Residuals for Test Set with Monitoring

Figure 4.1: Example Results of Monitoring an Intelligent Time Series Forecasting Model with a Tracking Signal [DJI04]

Gorr and McKay [GM05] are using a tracking signal based time series forecast monitoring to enable an automated recognition of crime pattern changes. They report that without automation, crime analysts have to visually inspect about 1000 time series plots of about five years length each month for medium-sized cities resulting in an unacceptably large workload.

When inspecting a crime related time series, "Analysts have to account for regular noise versus departures from established time trend patterns such as a sudden discrete change (step up or down) or a turning point (e.g., change from a decreasing time trend to an increasing trend.)" [GM05]. Based on the existing time series and their generated forecasts, for example locations and the sequence of police patrols are planned and when there is a deviation between the forecast and the most recent crime statistics is noticed, an adjustment of that planning might be required. As a forecasting model, they use a Holt exponential smoothing with smoothing parameters optimized [BO93] producing one-month-ahead forecasts.

To monitor the forecasts, they use the same tracking signal as Deng, Jaraiedi, and Iskander [DJI04] (see equation 2.11) with the parameters

- initial value for $E_t = 0$
- initial value for M_t is unknown
- $\alpha_1 = 0.4$
- $\alpha_2 = 0.05$ and
- the threshold for the tracking signal to alert as an experiment value.

Relating to McClain [McC88] who states "A perfect tracking signal would detect an out-of-control forecast (i.e., a time series pattern change) immediately, and would never give a false alarm.", Gorr and McKay follow a research design that questions the performance of the tracking signal as monitoring metric to detect signal changes while controlling for false positives. To achieve that they choose 10 crime time series including 5 having pattern changes and 5 not having any pattern changes. Gorr and McKay both independently manually marked each time series for pattern changes and outliers to generate a ground truth with 18 pattern changes or outliers in the 5 time series.

As a threshold for the tracking signal they used 4 values where the lowest value (0.84) detects most of the actual positives but also produces false alarms, so signals when a step change is found but was not labeled as such. As the other extreme, the threshold of 1.47, finds fewer actual positives but produces a lot less false positives. Between those, the values of 1.05 and 1.26 were used. In their research, they had one-month-ahead forecasts for a 36 month period excluding the first 6 months for burn-in purposes of the tracking signal resulting in a 30 month evaluation period of 10 time series. A crime analyst would have to assess these 10 time series every month, cumulating to 300 assessments over 30 months. The results (table 4.1) show the control limit as the threshold of the tracking signal, the amount of true positives detected with that limit, the average false positive rate per month and the workload the analyst would still have to do when evaluating only the time series where a trigger event took place. With the reported control limits, the workload can be reduced from 10 time series per month to 4 per month at a control limit of 0.84 and 94% of true positives detected. Together with

Control Limit	True Positives Detected	Average Workload (Time Series/Month)	Average False Positives (Time Series/Month)
0.84	17 (94%)	4.0	2.9
1.05	13 (72%)	2.8	1.9
1.26	12 (67%)	2.1	1.4
1.47	11 (61%)	1.6	1.0

Table 4.1: Results of Tracking Signal Control Limits and their Detection Rates [GM05]

the other results, it has been shown that with the usage of a tracking-signal-based monitoring, a significant decrease in workload for crime analysts can be reached. A challenge of this research design is that there needs to be a manual labeling of all evaluated time series and there is no certainty, that the labeling participants correctly and consistently label actual step jumps or outliers. Depending on the data, there might also be sector-specific knowledge required to correctly label the data.

4.2 ALTERNATIVE MONITORING FRAMEWORKS

Within the problem field of time series forecast monitoring, Candela et al. [Can+21] suggest a framework where multiple time series forecasting models for a single time series are maintained in parallel and another model built on top of that is charged with model performance monitoring and model selection.

The data used are 18.000 time series from the travel industry branch where one time series is the price of a flight to and a two day stay at a hotel in a specific city on a worldwide scale. Additionally they test their framework on the yearly and weekly M4 competition datasets [MSA18b]. They report that it has become relatively cheap to develop a highly accurate statistical model for time series price forecasting but the deployed model in production generates most of the costs and difficulties [Ré+19; Scu+15] since monitoring, maintenance and improvement need to be handled.

Candela et al. [Can+21] "[...] introduce a data-driven framework to continuously monitor and maintain time-series forecasting models' performance in-production, i.e in the absence of ground truth, to guarantee continuous accurate performance of travel products price forecasting models". In other words, they predict a models forecasting error for several models and by doing that, they can continously select the best performing model based on the predicted forecasting error without the requirement to wait for the future ground truth data.

The models generating the actual forecasts are called *monitored models* and the framework around that producing the predicted forecast errors is called *model selection framework* (fig. 4.2) with its monitoring models. The monitored models are provided multiple time series $\mathcal{X}$ with each several observations.

They produce forecasts and with the corresponding true values, the forecasting performance e_g can be calculated. The model selection framework respectively the monitoring models learn a function f to forecast the forecasting performance e_g of the monitored models directly from observations $\mathcal{X}$. When having learned f, the monitoring models can predict e_g^* given $\mathcal{X}^*$.

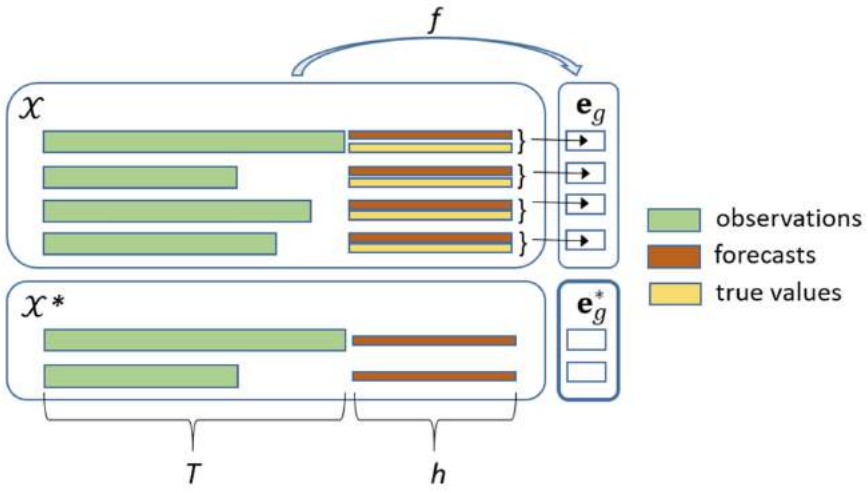


Figure 4.2: Model Selection Framework [Can+21]

To evaluate the forecasting performance, the Symmetrical Mean Absolute Percentage Error (SMAPE) [CY04] is used which is based on the MAPE (see equation 2.9). The known weaknesses of the MAPE that also apply with the SMAPE when observed values or forecasts are zero or close to zero, are no factor because the prices from the applied data are higher then in this problematic area.

The six monitored models used in their framework consist of five models from the 10 benchmarks provided in the M4 competition [MSA18b]: Simple Exponential Smoothing (ses), Holt's Exponential Smoothing (holt), Dampened Exponential Smoothing (damp), Theta and a combination of ses-holt-damp (comb). Additionally, a Random Forest in monitored.

For the model selection framework, four models from the latest advances in deep learning are tested:

- LSTM [HS97], which is a type of Recurrent Neural Network, that is equipped with several additional gates to solve the issue of the vanishing gradient problem [BSF94]. An advantage of LSTMs is that they can handle sequences of varying length.
- Convolutional Neural Networks (CNNs) [LeC+98], which are neural networks that are mostly used in image recognition and are used here with 1D convolutional filters to work with time series and are based on the LeNet [LeC+99] architecture.
- Bayesian Convolutional Neural Networks [GG16] as probabilistic CNNs that can quantify the uncertainty of predictions.

- Gaussian Processes [WR06], which are probabilistic nonparametric models that can also quantify uncertainty but can do that regardless of the size of the data.

The Gaussian Process model is found to be best performing as monitoring model. When a given monitored model falls below a set performance (SMAPE) threshold, a model selection process is started where the expected forecast error for every monitored model is evaluated and the model with the lowest error forecast is placed as the new production model. With this framework, quick indications of a predictive performance decrease can be given since there are no new ground truth data required. The best performing monitored model can always be selected. By doing this, the moment when a monitored model needs to be retrained could be extended by selecting another model as a first reaction. This could save costs when compared to more frequent retraining. Candela et al. [Can+21] do not experiment with model retrainings but limit the experiment to the model selection process.

Cohen, Garman, and Gorr [CGG09] have the understanding that "Time series monitoring methods have the purpose of detecting structural breaks or other unexpected pattern changes in time series data reliably and as soon as possible after those changes occur." which results in a binary classification problem - a pattern change is observed or not. They criticize the common approach used e.g. by McClain [McC88] to evaluate different methods and parameters of monitoring metrics, including tracking signals, by using "[...] idealized, simulated time series data and the average run length (ARL)." [CGG09]. The average run length is the number of periods between a chosen period and a signal trip where the average run length is desired "to be large for time series undergoing no pattern changes, and small (or timely) after a pattern change (or signal)." [CGG09]. In other words the average run length is the expected time for a monitoring metric to generate an alert with a given time series. For a time series with frequent unforeseeable pattern changes, the ARL should be small since an alert is expected. Cohen, Garman, and Gorr [CGG09] report that this results in a trade-off between quickly and reliably detecting a signal change and generating false alerts which is not discussed in literature. To address this and to give an alternate to the ARL statistic using simulated data, a Reciever Operating Characteristic (ROC) curve analysis using real data is introduced.

The major problem with establishing a ROC framework is that ground truth values are required that label pattern changes [Swe88]. To supply the required data, crime analysts labeled 30 time series from 606 available time series of crime data in the USA between 1991 and 2000. Predictions are generated with a deseasonalized simple exponential smoothing model one-month-ahead for 60 months, where the first 6 months are used as burn-in period and as historical information for the analysts. From the 30 selected time series, 10 were randomly chosen and 20 were chosen by the authors with the requirement of at least one potential step jump. In these 1620 months to eval-

uate (30 time series ∗ 54 months), 44 step jumps and 79 outliers were labeled. An example of step jump and outlier labeling is given in fig. 4.3.

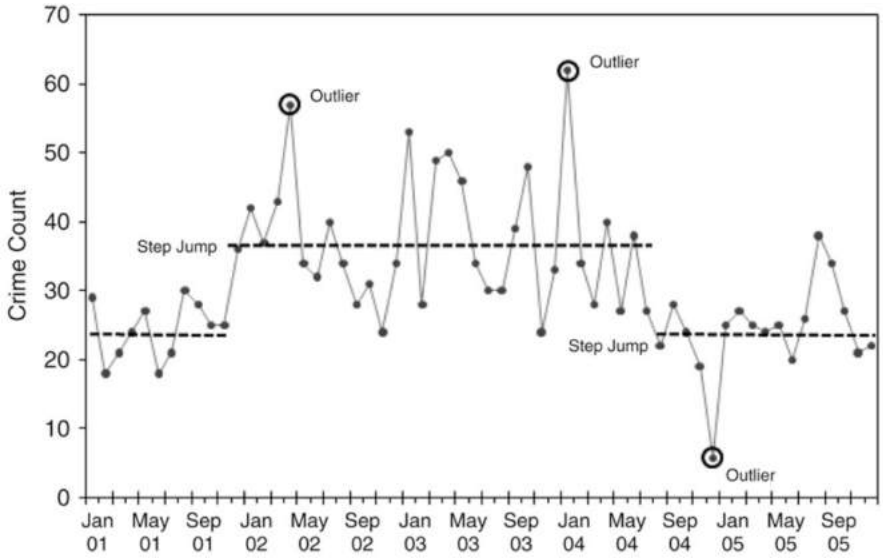


Figure 4.3: Example Time Series Coded for Pattern Changes [CGG09]

Five methods and their parameters used for monitoring time series forecasts are evaluated with the ROC framework: the Brown Method, the Trigg Method (which both are tracking signals), Standardized Forecast Errors (which is also based on one step-ahead-forecast errors using exponential smoothing), Percent Change and Standardized Observed value. As benchmark, random classification and a random classification with a four month detection window is given. Due to the high similarity in results between the Brown and the Trigg Method, the Brown method is not displayed [CGG09].

The comparison of the ROC curves (fig. 4.4) shows that the tracking signals (Brown and Trigg methods) perform best in the trade-off of True Positive and False Positive Rate, followed by Standardized Forecast Errors, Standardized Observations and the Percent Change. This is limited to the observed crime data and the reliability of the ground truth coding though. For other domains with different data, this assessment would have to be evaluated again but in general, the ROC framework seems to be a valid alternative to the evaluation of monitoring metrics and their parameters to reach optimal monitoring performance with the help of ARL. It should be emphasized again though that ground truth data for pattern changes is required for the ROC framework. As another result, the authors find that when using a tracking signal, the most recent forecast error should be used as the numerator and for the denominator, the MAE of forecast errors should be smoothed with a relatively low smoothing constant.

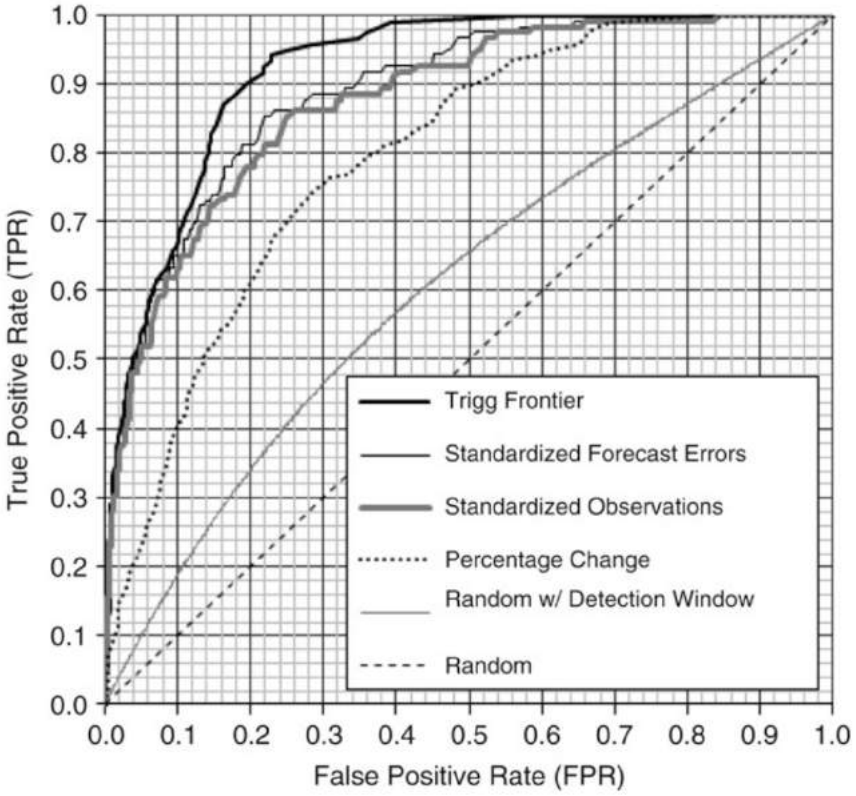


Figure 4.4: ROC Curves for Alternative Time Series Monitoring Methods, Only Trigg Tracking Signal (Trigg Frontier) Displayed, Tracking Signals with best Performance [CGG09]

5 OWN EMPIRICAL ANALYSIS AND RESULTS

To answer the questions and hypothesis raised in chapter 3, the conducted experiment of the benchmarking between the AWS default model quality monitoring and a tracking signal is described here. The goal of the experiment is to evaluate and potentially optimize the cost-benefit relation of costs raised in a time series forecasting MLOps environment. First, the two datasets of time series with hourly and daily interval are presented including the random selection of sample data. Secondly, the monitored models are described as well as the methodology of how the benchmarking is established. Finally, the results of both datasets are presented and then discussed in the following chapter 6.

5.1 DATA

Time series forecasting competitions have been an essential part in the research advancement of time series forecasting for about 50 years since computers became widely available [Hyn20]. After a controversial start, nowadays these competitions are well established and make important contributions to the state-of-the-art research. Especially the M-competition series which was first published in 1982 by Makridakis et al. [Mak+82] had a revolutionary impact on time series forecasting [Hyn20]. Over the years, this competition series was continued with the latest results from the M5 competition [MSA22] and a running M6 competition until the end of 2023 with prizes in worth over US$ 300.000 [Mak22; For22]. Besides the results of the competitions, the published datasets for training and testing represent benchmark datasets for the time series forecasting community used in several experiments.

The experiment conducted here takes advantage of the data provided in the M4 competition [MSA18b] (see fig. 5.1).

Time interval between successive observations	Micro	Industry	Macro	Finance	Demographic	Other	Total
Yearly	6,538	3,716	3,903	6,519	1,088	1,236	**23,000**
Quarterly	6,020	4,637	5,315	5,305	1,858	865	**24,000**
Monthly	10,975	10,017	10,016	10,987	5,728	277	**48,000**
Weekly	112	6	41	164	24	12	**359**
Daily	1,476	422	127	1,559	10	633	**4,227**
Hourly	0	0	0	0	0	414	**414**
Total	**25,121**	**18,798**	**19,402**	**24,534**	**8,708**	**3,437**	**100,000**

Figure 5.1: Number of M4 Time Series per Frequency and Domain [MSA18b]

100.000 time series divided over six different datasets/observation intervals (yearly, quarterly, monthly, weekly, daily, hourly) are provided over 6 domains. The randomly sampled data originates from the ForeDeCk database

Daily Index	Daily Observations	Hourly Index	Hourly Observations
151	954	3	700
481	4196	25	700
558	2940	39	700
729	4197	156	700
2231	4196	165	700
2416	3444	222	960
2792	4197	274	960
2950	4197	334	960
3106	4197	343	960
3129	4197	354	960

Table 5.1: Daily and Hourly Sample Data

compiled at the National Technical University of Athens (NTUA) containing 900.000 continuous time series, built from multiple, diverse and publicly accessible sources from several domains like industry, services, tourism, imports & exports, demographics, education, labor & wage, government, households, bonds, stocks, insurances, loans, real estate, transportation, and natural resources & environment [Spi+20].

Two of the six available datasets are used in the conducted experiment because a realistic scenario is expected in an MLOps environment with **daily** and **hourly** frequencies. Also, the time series in these two datasets (daily & hourly) show different characteristics which adds some robustness to the results of the research. There is no differentiation between originating domains. For each dataset, 10 non-intermittent time series were randomly selected, which means the observation count can vary between the time series but there are no missing values within one time series (see table 5.1).

5.1.1 *Daily Dataset*

In the original M4 competition, the forecast horizon, so the number of forecasts into the future which are used for model evaluation, is 14 one-step-ahead predictions for the daily dataset. The participants were provided the training dataset and after completion of the period to send in solutions, a test dataset with 14 observations was provided and used to compare the submitted solutions. In our research, a much longer monitoring horizon is chosen of 100 forecasts to give a more realistic assessment of time series forecasting in an MLOps environment where retraining the model is planned. The forecast horizon varies depending on the time of retraining. The test dataset is ignored and the last 100 observations of the sampled time series from the daily dataset are used as test data. The size of the training time series varies between 854 and 4097 observations (table 5.1). This length is larger then in

previous competitions like the M3 competition in order to avoid overfitting and enable the generation of simple and complicated models that require large amounts of training data [MSA20]. The monthly and especially the yearly and quarterly datasets show much shorter lengths compared the daily and hourly datasets. As a minimum length of a time series, 600 observations are set for the sampling before splitting into test/training data to guarantee a good relation between the amount of training to test data. The time series are all non-stationary with changing trends and no clear seasonality. For an example time series see fig. 5.2, for the complete 10 sampled time series see appendix chapter A.1. The last 100 observations as test data are colored in orange.

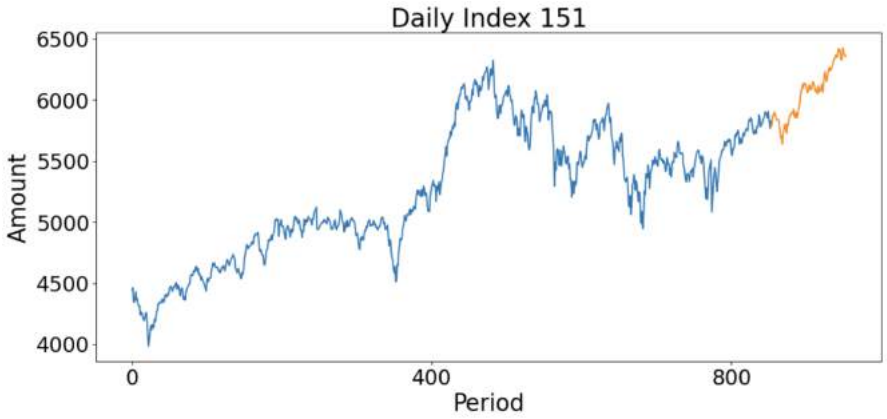


Figure 5.2: One Random Sample of Daily M4 Dataset with Training Data and 100 Test Data Points

5.1.2 *Hourly Dataset*

For the hourly dataset, the M4 competition defined a forecast horizon of 48 one-step-ahead predictions whereas we use a monitoring horizon of 100 like with the daily dataset. The forecast horizon varies depending on the time of retraining. Compared to the daily dataset, the hourly data is more uniform with more clear seasonality and less changing trends. For an example time series see fig. 5.3, for the complete 10 sampled time series see appendix chapter A.2.

5.2 MODELS AND METHODOLOGY

The monitored models in the research conducted here are chosen to represent widely used models for time series forecasting. The focus is laid on the evaluation of the number and timings of model retrainings depending on different monitoring approaches. The models used are not thoroughly optimized to give the best possible forecast like it is the goal in the M competition series where mostly models consisting of multiple models or even

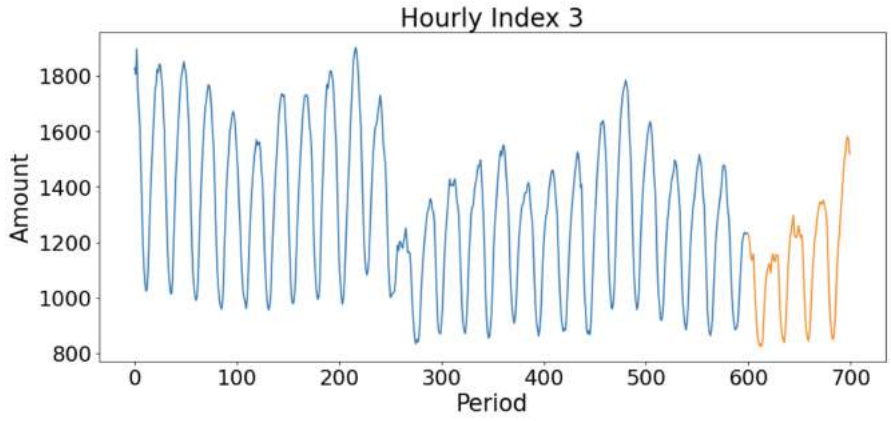


Figure 5.3: One Random Sample of Hourly M4 Dataset with Training Data and 100 Test Data Points

hybrid models score best. This results in two models being evaluated here: ARIMA and ETS (see chapter 2.2.2.1).

To implement the ARIMA models, the Auto-ARIMA library from pmdarima [1] is used in a python environment. To build a model, training data is used to generate multiple ARIMA models with different model parameters. When the search with all parameter combinations within the search space is completed, the model with the lowest AIC is chosen. The default options for the Auto-ARIMA process are selected. An explorative search with other parameters did not provide better results at longer evaluation times.

The ETS models utilize the statsmodels ETS library [2] where the parameters seasonal and trend are added and true damped trend is used. Other parameters remain on default.

For every time series, the evaluation procedure is identical and can be listed as followed:

1. Isolate single time series from dataset
2. Preprocess data (drop null values at the end of the time series)
3. Generate test/training split
4. Further process data for specific model
5. Build model on training data
6. Evaluate performance on training data (in-sample predictions), generate baseline
7. Produce forecasts for complete monitoring horizon without retraining (forecast horizon = 100), evaluate performance

1 https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html
2 https://www.statsmodels.org/devel/examples/notebooks/generated/ets.html

8. Produce forecasts with retraining at every new observation (forecast horizon = 1), evaluate performance
9. Produce forecasts with AWS default model monitoring method, retrain at signal (variable forecast horizon), evaluate performance
10. Produce forecasts with tracking signal method and 2 thresholds, retrain at signal (variable forecast horizon), evaluate performance
11. compare results regarding costs and performance

When using the AWS model monitoring method, only the first three of the available monitoring metrics (MAE, MSE, RMSE and R^2) are implemented in the experiment described here because the R^2 is typically not used in time series monitoring (e.g. [DF13; DF16]) due to a changing mean and variance over time which can distort the interpretability of the R^2. The R^2 is often used in other regression problems to explain the influence of the independent variables on the variance of the dependent variable.

Since the default thresholds for the AWS monitoring metrics are the errors of the in-sample prediction, the error values of MAE, MSE and RMSE for the in-sample prediction are used as such monitoring thresholds.

The AWS monitoring is implemented with a monitoring schedule every 3 observations, which can be translated to every 3 days or every 3 hours, depending on the dataset. Within this window, the three monitoring metrics MAE, MSE and RMSE are calculated for the last 3 observations and their ground truth data. At a first step, forecasts for the whole monitoring horizon are generated without any retraining. Then this 3-observation-window is slided over the forecasts and the error measures are calculated. When one of the monitoring thresholds are exceeded, a retraining signal is given, all observations to this point are provided to the model and a retraining takes place. After the retraining, the remaining forecasts to the end of the monitoring horizon are generated. Then the next iteration of evaluation with the sliding window, retraining and forecast generation takes place. This happens until the end of the monitoring horizon is reached. The position of the retraining points and the count of the different threshold violations are saved. After this process, the error measures are evaluated over the whole monitoring horizon.

As parameters for the tracking signal (see equation 2.11) we set based on the recommendations from related research [GM05; CGG09; McC88]

- the initial value for $E_t = 0$
- the initial value for $M_t = 1$
- $\alpha_1 = 0.4$
- $\alpha_2 = 0.05$

- the threshold for the tracking signal to alert at 1.2 and as an experiment value so that the number of retrainings in this adapted setup is comparable to the number of retrainings that are triggered when using the AWS monitoring method.

With the tracking signal monitoring, at first again 100 forecasts for the whole monitoring horizon are generated. After a burn-in phase of 5 observations, the calculated tracking signal is compared with the provided threshold and when it is exceeded, all observations to this point are provided to the model and a retraining takes place after which further forecasts are generated. With the additional forecasts, another iteration of tracking signal evaluation, threshold exceedance, retraining and forecast generation takes place until the end of the monitoring horizon is reached. The position of the retraining points are saved and the error measures MAE, MSE, and RMSE are evaluated over the whole monitoring horizon.

Besides the tracking signal threshold of 1.2, a tracking signal value of 0.4 is found to generate a count of retrainings over the monitoring horizon, which can be compared to the count of retrainings produced by the AWS monitoring method. With this value, the process of the tracking signal monitoring is repeated.

To evaluate the costs of the process, the prices of AWS EC2 instances appropriate for this computation are used. For m5.xlarge instances with 4 vcpus and 16gb of RAM, hourly costs of **US$ 0.23** apply [3]. To estimate the ecological footprint measured in CO_2, the hourly value of **13.3 g** is estimated [4]. Also, m5.xlarge instances are used for the calculation and the european frankfurt region is set. 50% CPU load is assumed at 28.2 Watts. The emissions generated at producing the computing instance is considered as well. All calculations are based on a public dataset [5] containing specifications of multiple computing instances. At the time of writing, the ecological costs are not to be compensated by the end user directly but there is a system of CO_2 certificate trading in place which forces companies to compensate their negative ecological footprint by buying these certificates [Kom22]. This puts a price on the CO_2 emission which might in the future be payed for directly by every user. With this price, a translation of the estimated grams of CO_2 emission to costs in € is computed. As base price, **€ 55** per ton of CO_2 is used, which will be effective from 2025 on [Bun22].

To compute prices for the financial and ecological aspect, the time required for the computation is needed as well. During the evaluation of the different monitoring approaches, the training timings of the models are measured and then multiplied with the hourly costs. Since we evaluate on a daily or hourly schedule, it is assumed, that the deployed model stays in production the whole time but separate instances for the retraining are activated only when required. We experience startup times of about 5 minutes with AWS to

3 https://aws.amazon.com/de/ec2/pricing/on-demand/

4 https://engineering.teads.com/sustainability/carbon-footprint-estimator-for-aws-instances/?estimation=true&instance_id=2425&region_id=2247&compute_hours=1#calculator

5 https://docs.google.com/spreadsheets/d/1DqYgQnEDLQVQm5acMAhLgHLD8xXCG9BIrk-_Nv6jF3k/edit#gid=50475527

prepare the training EC2 instance before the actual computation can begin so this time is added at each training period.

The decrease of the model performance is another factor that has to be implemented in the framework because depending on the business environment, a decreasing model performance can result in high costs which could have been reduced with a more frequent model training or the implementation of different model types or parameters. The resulting costs of the degrading model performance can heavily vary but for this research, we assume **€ 0.01** per error value per period. Therefore, the costs in € of the combination time series and monitoring method are calculated by MAE $*$ monitoring horizon $*$ 0.01.

As alternative experiment setup, the most recent observation could be supplied to the models in every case without starting a new training but rather update the existing model with the newest observation. This experiment would have a different character and assess when a reevaluation of the model parameters is required although the newest observation is supplied at every step. This alternative approach is not followed in this research because of several reasons. Firstly, this research orientates on related research like by Deng et al. [Den+09] to serve as benchmark. Secondly, the availability of such an update method with time series forecasting models is limited. Regarding the used models, the Auto-ARIMA library offers an update method to provide new observations without computing a retraining in the parameter search space but the ETS model has no equivalent function. This could be the case because of the fast training time with the ETS model that makes an update function obsolete. Finally, in an MLOps environment as examined in this research, the set goal is to reduce costs by limiting the times a computing instance has to start to alter the model. In the alternative scenario, an update or a retraining would take place at each step along the time series, requiring to regularly start a computing instance that generates costs. We want to examine ways to reduce the times, computing instances are required to alter the model, so an update of the model with every observation is not adequate in this setup.

To report the process of the experiment even more transparent, for each dataset with one time series, the journey through the evaluation framework is explained in the following two subchapters. To present this more compact, the two model types (ARIMA, ETS) are described in parallel. In the actual experiment, the model types are build and evaluated sequentially.

5.2.1 *Example Time Series of Daily Data*

In the following, we describe the evaluation process with an isolated time series from the daily M4 dataset with the index 151 displayed in fig. 5.2. At this point, the time series is already preprocessed (all null values after the last observation are dropped) and the test/training split is executed (the last 100 observations of the time series are declared as test data and the first 854 observations become training data). After that, further processing for the

ETS model is required since the model library requires a series consisting of values as numpy array and the observation time or index as date range.

In an iterative process, the ARIMA and the ETS models are build with the model parameters being evaluated automatically by minimizing AIC or maximizing log likelihood [HA18]. The process of the Auto-ARIMA including the final model parameters is displayed in fig. 5.4. The model parameters tested can be viewed in the first column.

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=9307.515, Time=0.64 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=9313.224, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=9313.941, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=9313.828, Time=0.09 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=9311.926, Time=0.01 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=9313.275, Time=0.36 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=9312.792, Time=0.27 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=9309.050, Time=0.77 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=9310.380, Time=0.70 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=9313.389, Time=0.35 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=9313.725, Time=0.32 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=9312.652, Time=0.46 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=9308.962, Time=0.81 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=9306.028, Time=0.35 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=9312.090, Time=0.10 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=9311.608, Time=0.08 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=9307.766, Time=0.36 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=9309.093, Time=0.24 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=9312.688, Time=0.13 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=9312.436, Time=0.15 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=9311.379, Time=0.08 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=9308.171, Time=0.53 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
```

Figure 5.4: Iterative Auto-ARIMA Process - Daily Example

With the given model, the in-sample error metrics MAE, MSE and RMSE are calculated, which later serve as threshold for the AWS monitoring approach. The plot of in-sample forecasts against training data for the ARIMA model is displayed in fig. 5.5. The plot for the ETS model is very similar and can be found in the appendix as fig. A.3. Both in-sample forecast stay close to the actual values, following the level and trend of the last observations resulting in a small lag of the forecast curve after the ground truth curve. To present this more clearly, only the last 100 forecasts and ground truth values of the ARIMA model are displayed in fig. 5.6. The ETS model looks almost similar showing the same offset character of the forecast curve behind the ground truth curve and can be found in the appendix in fig. A.4. The error measures for the in-sample prediction are slightly lower for the ARIMA model for this time series but the two model types perform on a similar level.

The results for the in-sample error measures used as AWS monitoring thresholds are listed in table 5.2. All metrics function as *greater then* threshold meaning that if in later comparisons, values greater then the thresholds

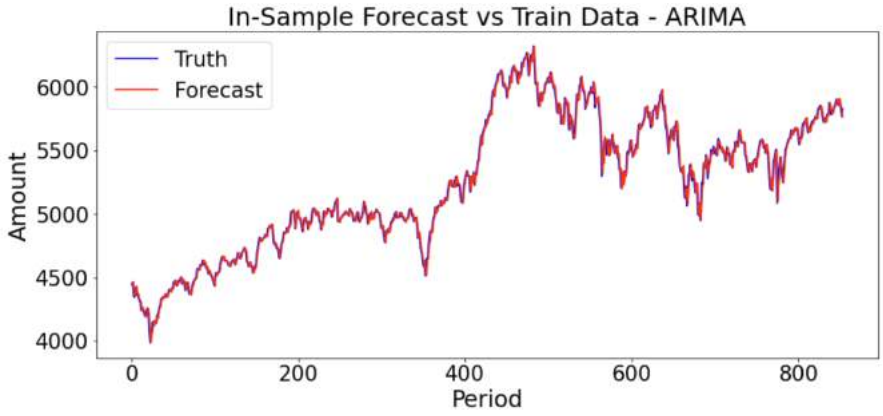


Figure 5.5: In-Sample Forecasts vs Training Data for ARIMA Model - Daily Example

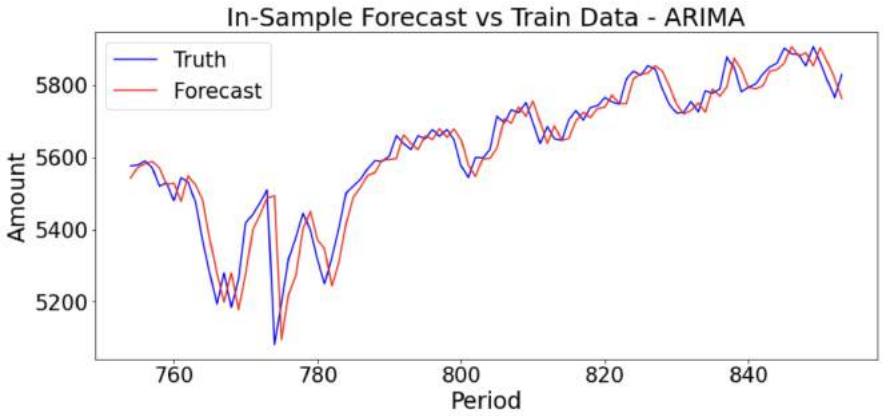


Figure 5.6: In-Sample Forecasts vs Training Data for ARIMA Model, last 100 Observations - Daily Example

Metric	ARIMA	ETS	Threshold
MAE	40.89	40.91	greater then
MSE	3162.09	3179.99	greater then
RMSE	56.23	56.39	greater then

Table 5.2: In-Sample Error Measures - Daily Example

are monitored, an alert is produced that results in a model retraining with the most recent available ground truth data.

With the two models at hand, predictions for the complete monitoring horizon are generated, so a forecast horizon of 100 is set. Plots for ARIMA and ETS models are in figure 5.7 and 5.8. In both figures only the last 100 observations of the training data are displayed to focus on the forecasts but also provide reference to the latest data before the forecasting period.

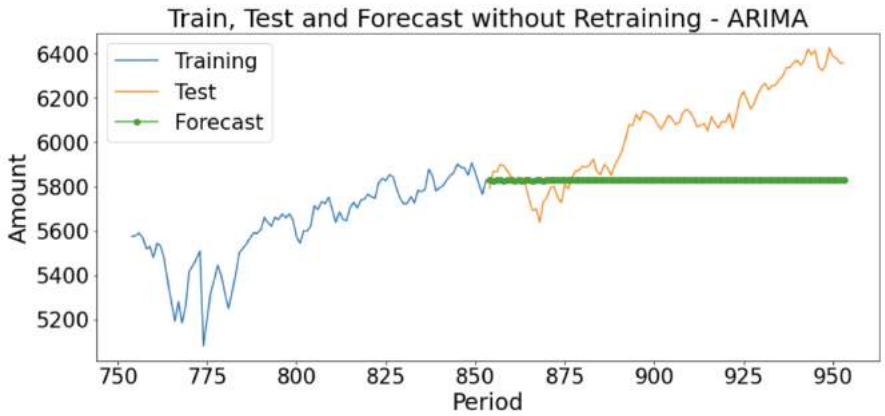


Figure 5.7: In-Sample Forecasts vs Training Data for ARIMA Model - Daily Example

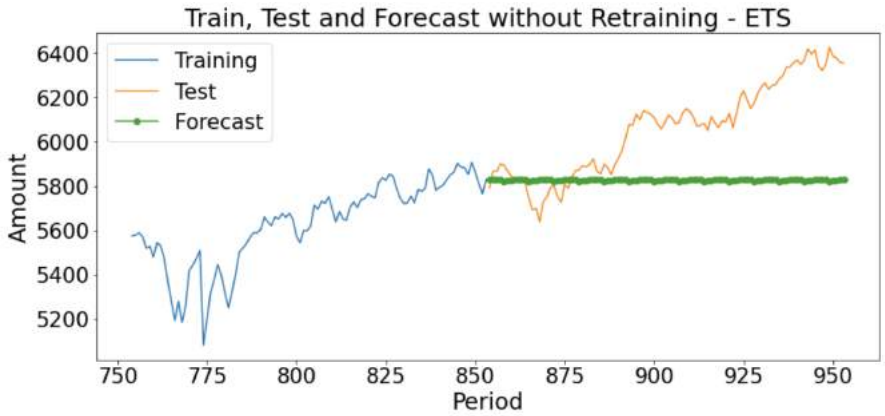


Figure 5.8: In-Sample Forecasts vs Training Data for ARIMA Model - Daily Example

It can be observed that with both models, the forecasts heavily depend on the latest observation and extrapolate on that level with a minor seasonal component, which is more visible for the ETS model. This is not an inherent phenomenon of the model types but is an effect of the volatile time series

that does not follow any clear trend or seasonality. To explore the following monitoring also with a different forecasting behaviour, the hourly M4 dataset is analyzed as well.

The other extreme besides generating forecasts with the same model for the complete monitoring horizon is to always generate a single forecast and then retrain the model with the newest observation. This approach generates the highest training costs but should keep the forecasting error on a minimum since changes in the time series can very quickly be adapted to. Charts for ARIMA and ETS models are in fig. 5.9 and appendix fig. A.5. Again, only the last 100 observations of the training data are displayed. As with the in-sample forecasts, a short lag of the forecast curve behind the ground truth data is visible due to a relatively high weight of the forecast on the last observation and the supply of the newest ground truth observation at every period. The error evaluation is displayed in table 5.3 and shows a almost similar performance for ARIMA and ETS model.

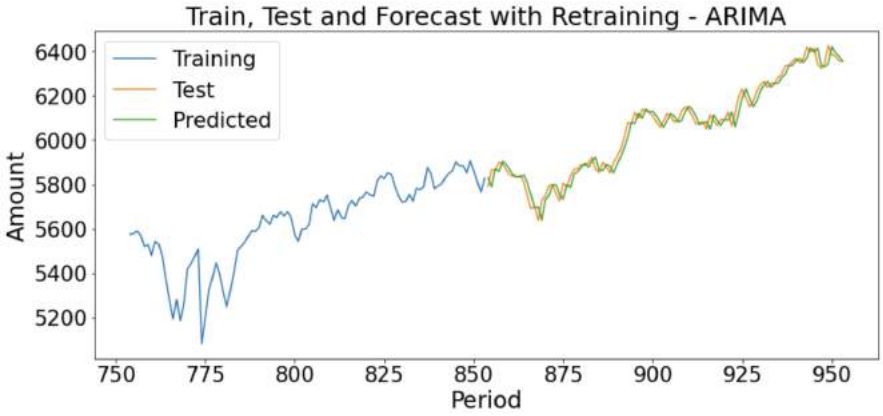


Figure 5.9: Always Retrain Forecasts vs Training Data for ARIMA Model - Daily Example

When using the **AWS monitoring** method, 33 retrainings are executed with the ARIMA model (table 5.3) due to exceedance of the baseline thresholds. In all cases, the MAE triggers an alarm and in 20 of the cases each, MSE and RMSE thresholds are exceeded. For the ETS model, 34 retrainings are executed where the MAE also is the dominating signal-generating error measure. The plot of the last 100 training observations, the forecasts, the test data and the retrainings generated by the AWS monitoring can be found for ARIMA and ETS model in fig. 5.10 and appendix fig. A.6. In both figures, it can be observed that a step change or trend change like around periods 870, 890 and 945 results in a higher count of retrainings there since the model's need to adapt to the new level or structure of the recent observations. The error evaluation is displayed in table 5.3 showing the slightly better performance of the ETS model over the ARIMA model which could be connected to the extra retraining for the ETS model. The extra retraining results in slightly higher training costs but lower total costs can be observed due to the reduced costs from the error induced by the model performance. The extra retraining also results in a slightly greater ecological footprint and resulting CO_2 costs.

For a reference of the MAE over time impacted by the retrainings, a plot of the error compared to the baseline of the MAE can be seen in fig. 5.11 for the ARIMA model and in the appendix at fig. A.7 for the ETS model.

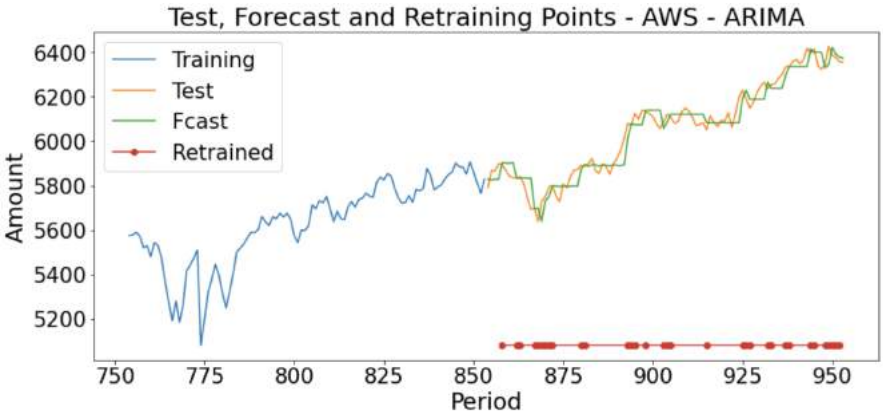


Figure 5.10: AWS Retrain ARIMA Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Daily Example

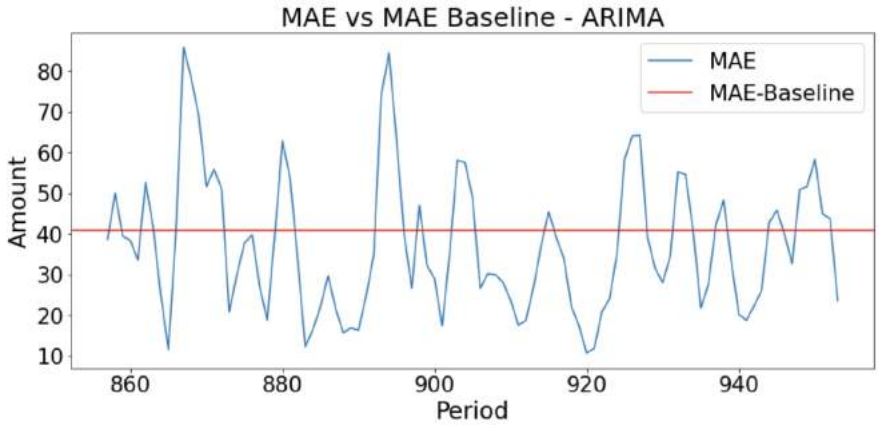


Figure 5.11: MAE Over Time - ARIMA - Daily Example

With the **tracking signal monitoring** and a threshold of 1.2 the ARIMA model is retrained 22 times and the ETS model 23 times (see table 5.3). Although one less retraining is computed, the ARIMA model shows lower error measures. Due to the lower retraining amount and a better performance, the ARIMA model with the tracking signal monitoring and a threshold of 1.2 has better cost values then the ETS model with the same monitoring in every aspect including the ecological costs (see table 5.3. Plots for ARIMA and ETS models can be found in fig. 5.12 and appendix fig. A.8. The timings of the retraining are very similar with the most prominent difference at the end of the time series, where the ETS model has an additional retraining.

In an explorative search, the **tracking signal threshold of 0.4** is found to result in comparable retraining counts to the AWS method. This particular time series is an outlier in that relationship though (compare table 5.4) resulting for the ARIMA model in 68 retrainings against 33 with the AWS monitoring and the ETS model 68 compared to 34 AWS retrainings. The results presented

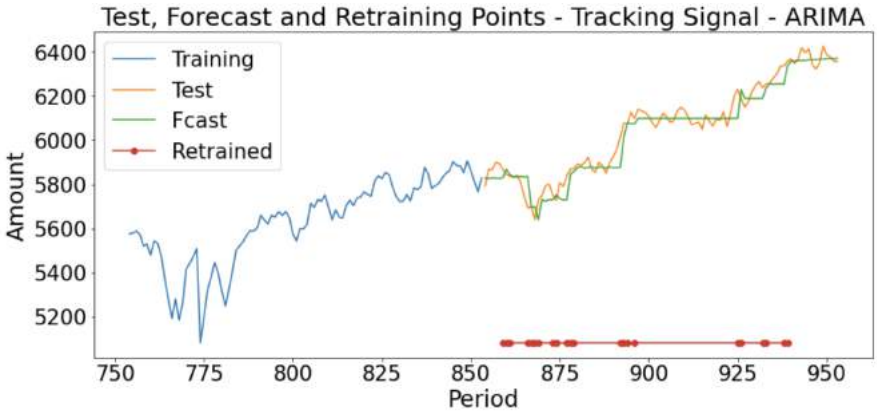


Figure 5.12: Tracking Signal Monitoring ARIMA Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Daily Example

(see table 5.3) show reduced error metrics of the adapted tracking signal monitoring when compared to the AWS monitoring and the tracking signal monitoring with a threshold of 1.2. With a tripled retraining count compared to the higher tracking signal threshold and a doubled count compared to the AWS monitoring, the errors are still about 20% higher then the always retrain errors. At this error level, lower retraining counts are desirable to reduce economical and ecological costs. For plots of the monitoring with the adapted tracking signal see appendix fig. A.9 and A.10.

This walk through the evaluation framework is supposed to give a better understanding of the way, results are computed. The shown data is only for one time series and has thus limited explanatory power. In this scope, the ARIMA model with the always retrain method shows the lowest total costs since the costs of the decreasing model performance is about 16 times the cost of the retraining. It should be emphasized that especially the costs of the error (decreasing model performance) is extremely variable with the specific usecase. Furthermore, the costs of the retraining are highly dependent on the startup time of the cloud computing instance. With the following subchapter, another example with a time series from the daily dataset is given but a thorough discussion of the research and its results will take place in chapter 6 after the presentation of the overall results.

	N Retrain	MAE	MSE	RMSE	Cost Training	Cost Error	Total Costs	Co2 [g]	Co2 [€]
In-Sample ARIMA		40.89	3162.09	56.23					
In-Sample ETS		40.91	3179.99	56.39					
Never Retrain ARIMA	0	253.00	96592.34	310.79	0.00	253.00	253.00	0.00	0.0000
Never Retrain ETS	0	252.98	96693.71	310.96	0.00	252.98	252.98	0.00	0.0000
Always Retrain ARIMA	100	**29.47**	**1295.07**	**35.99**	1.85	29.47	**31.32**	**112.65**	0.0062
Always Retrain ETS	100	31.16	1388.51	37.26	1.82	31.16	32.98	110.98	0.0061
AWS Retrain ARIMA	33	36.91	2141.37	46.27	0.61	36.91	37.52	37.18	0.0020
AWS Retrain ETS	34	35.71	2023.02	44.98	0.62	35.71	36.33	37.73	0.0021
TS ARIMA	22	36.75	2249.36	47.43	0.41	36.75	37.16	24.78	0.0014
TS ETS	23	38.34	2368.80	48.67	0.42	38.34	38.76	25.52	0.0014
TS Adapted ARIMA	68	33.60	1686.40	41.07	1.26	33.60	34.86	76.68	0.0042
TS Adapted ETS	68	35.33	1829.37	42.77	1.24	35.33	36.57	75.45	0.0041

Table 5.3: Results of Daily Example Time Series, all Costs in €, TS = Tracking Signal, Total Cost = Cost Training + Cost Error

5.2.2 *Example Time Series of Hourly Data*

In this subchapter the evaluation process of the time series with index 3 from the M4 hourly dataset (shown in fig. 5.3) is described. The approach is the same as with the presented daily example time series. Here, only new aspects resulting from the different dataset are presented. The Auto-ARIMA process for the basic ARIMA model is displayed in appendix fig. A.11.

The in-sample forecasts are both very close to the training data and can be found in the appendix as fig. A.12 and A.13. AWS monitoring thresholds from the in-sample errors and results from the example time series are displayed in table A.1.

The forecasts without retraining for ARIMA and ETS model displayed in fig. 5.13 and 5.14 show very different results with the ARIMA model following the most recent seasonal pattern and the general downward trend whereas the ETS model generates almost duplicates of the last known value. This extreme behaviour is not observed with other daily time series (compare appendix fig. A.14 or A.15 for two other time series from daily dataset). In other cases the existing pattern is followed with the ETS model, but in a much weaker manner then with the ARIMA model.

The monitoring approach of always retraining the model with the newest data point shows no difference to the daily dataset for which reason an additional display is omitted. The AWS monitoring generates 68 retrainings for the ARIMA model and 65 for the ETS model. Due to the very good in-sample fit, the forecasts very rarely are better then the thresholds from the in-sample forecasts and a lot of alerts are generated as a result (see fig. 5.15 and A.16).

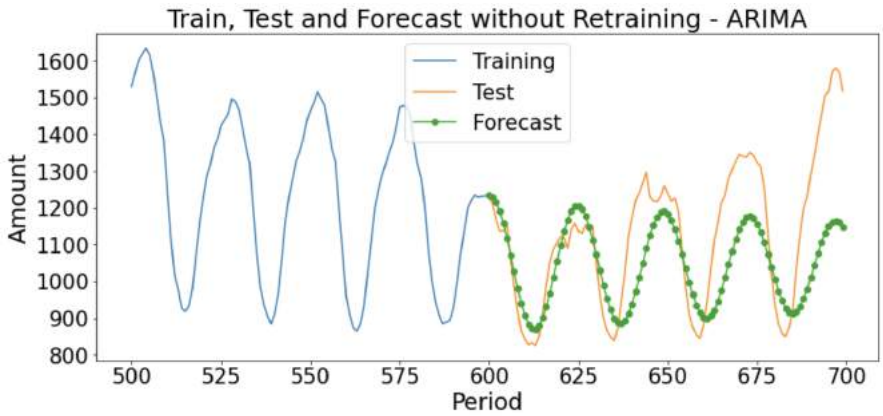


Figure 5.13: In-Sample Forecasts vs Training Data for ARIMA Model - Hourly Example

With the tracking signal monitoring and the threshold of 1.2, for the ARIMA model 32 and for the ETS model 36 retrainings (see table A.1) are triggered. This is a substantial reduction compared to the AWS monitoring, which comes with much larger error measures on the other hand. With the given costs for the decreasing model performance, the AWS monitoring results in

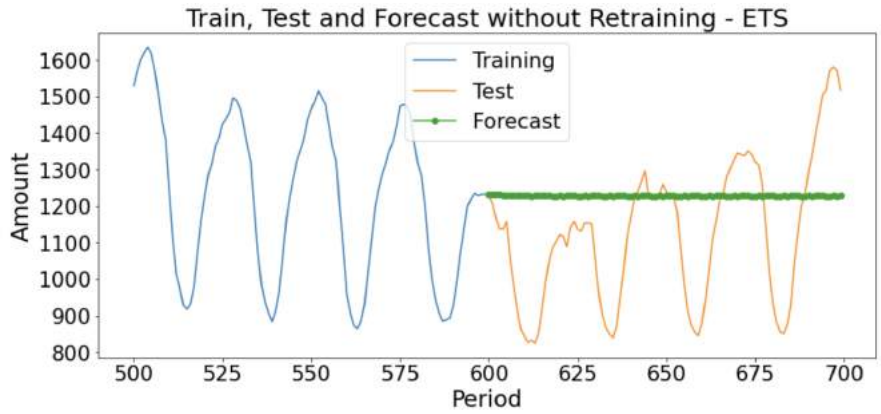


Figure 5.14: In-Sample Forecasts vs Training Data for ETS Model - Hourly Example

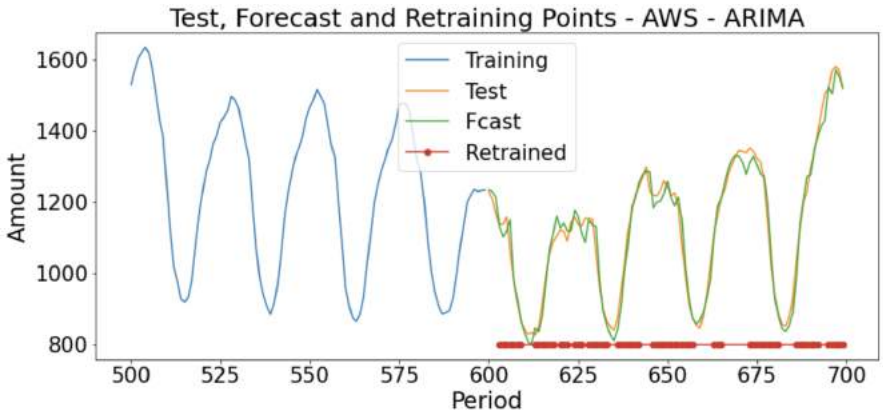


Figure 5.15: AWS Retrain ARIMA Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Hourly Example

lower total costs, but a higher environmental impact due to the higher retraining count.

The adapted tracking signal enables the comparison between 68 AWS retrainings and 64 with the adapted tracking signal with the ARIMA model and 65 against 60 with the ETS model. With this time series, the retraining count is on a comparable level but the error metrics are much higher with the adapted tracking signal monitoring then with the AWS monitoring (see table A.1). This also results in higher total costs due to the decreases predictive performance.

Overviewing, the ARIMA model with the always retrain method has the lowest total costs with the given error costs but the highest environmental costs as well.

5.2.3 *Results Daily Data*

To give a more reliable understanding of the differences between the evaluated monitoring methods in combination with the monitored models, the results of the 10 sampled time series from each dataset are collected and presented as average results.

With the 10 time series from the daily M4 dataset (see table 5.4), the in-sample error measures of the two model types lead to similar AWS monitoring thresholds. The lowest total costs are generated by the ETS always retrain method but the highest environmental impact is generated by the ARIMA always retrain method since the Auto-ARIMA process used for every retraining is computationally more expansive then the automated ETS model training. Besides that, there are no major differences found between the two model types.

Compared to the CO_2 generated with the always retrain method, the AWS method almost cuts the amount in half. The tracking signal monitoring with the threshold of 1.2 produced about a quarter of the always retrain CO_2 but with fewer retrainings, the error measures are increasing as well. The MAE of the always retrain method is about 76 whereas the AWS monitoring method increases the MAE to about 81 and the tracking signal results in values around 120. If the trade-off of a reduced retraining count against an increased forecasting error favours fewer retrainings and a performance drop is accepted, the tracking signal is a fitting monitoring metric. With the value set in this research, the method of always retraining generates the lowest costs but also results in the highest environmental impact.

To never supply more recent observations to the model and adjust the forecast with this information is an option which can not be recommended when using this kind of monitoring horizon and time series since a massive error increase results. On the other hand, the environmental impact caused by updating the model is not existent.

The adapted tracking signal, compared to the AWS monitoring method, results in slightly increased retraining counts but fails to reduce the forecasting error at the same time. It seems like the timings of the retrainings triggered

by the AWS monitoring method are more fitting, probably closer to an actual change in the time series, to reduce the error of future forecasts.

5.2.4 *Results Hourly Data*

With the 10 time series from the M4 hourly dataset, the ETS model has a slightly better in-sample fit and lower always retrain error then the ARIMA model (see table 5.5). In all other scenarios, which depend on good forecasts on a forecast horizon greater then 1, the ARIMA model performs much better then the ETS model. This can be related to the different forecast patterns generated and already displayed in figures 5.13 and 5.14. With the knowledge of this relation, the ETS model with its selection process of the best parameters should be adjusted to generate better forecasts over a longer forecast horizon with this dataset.

The lower training costs of the ETS models are caused by a quicker training process. The highest environmental impact is, just like with the daily dataset, generated by the always retrain ARIMA model. The AWS monitoring method reduces the retrainings from 100 to about 60 (ARIMA) and 55 (ETS) which results in increased error metrics of about 13% compared to the always retrain method when looking at the ARIMA model and about 35% with the ETS model. The tracking signal with the threshold of 1.2 reduces the retraining count to about 30 but the error measures increase by about double of the always retrain value with the ARIMA model. Due to the bad performance of the ETS model with this dataset, the error measures are drastically increasing with a reduced retraining count.

The adapted tracking signal with the 0.4 threshold fails here, like with the daily dataset, to produce comparable or even better error metrics then the AWS method. At similar retrainings, especially the ETS model produces much higher errors.

	N Retrain	MAE	MSE	RMSE	Cost Training	Cost Error	Total Costs	Co2 [g]	Co2 [€]
In-Sample ARIMA		63.29	17679.67	109.31					
In-Sample ETS		63.74	17413.10	107.88					
Never Retrain ARIMA	0	551.98	979591.19	619.62	0.00	551.98	551.98	0.00	0.0000
Never Retrain ETS	0	539.35	949778.68	603.80	0.00	539.35	539.35	0.00	0.0000
Always Retrain ARIMA	100	76.56	13129.23	99.42	1.95	76.56	78.51	**118.9**	0.0066
Always Retrain ETS	100	75.84	13074.55	99.28	1.82	75.84	**77.66**	111.0	0.0061
AWS Retrain ARIMA	59.4	81.82	14717.50	106.24	1.16	81.82	82.98	70.7	0.0039
AWS Retrain ETS	58.1	80.71	14482.44	105.24	1.06	80.71	81.77	64.5	0.0036
TS ARIMA	23.9	122.89	34424.48	157.53	0.47	122.89	123.36	28.4	0.0016
TS ETS	24.4	117.17	31575.43	151.07	0.44	117.17	117.61	27.1	0.0015
TS Adapted ARIMA	62.5	92.22	19169.75	120.57	1.22	92.22	93.44	74.3	0.0041
TS Adapted ETS	60.7	93.05	21122.28	124.24	1.11	93.05	94.16	67.4	0.0037

Table 5.4: Results of 10 Daily Time Series, all Costs in €, TS = Tracking Signal, Total Cost = Cost Training + Cost Error

	N Retrain	MAE	MSE	RMSE	Cost Training	Cost Error	Total Costs	Co2 [g]	Co2 [€]
In-Sample ARIMA		156.21	335764.60	211.45					
In-Sample ETS		148.66	359972.28	222.66					
Never Retrain ARIMA	0	1060.16	14947306.75	1321.18	0.00	1060.16	1060.16	0.00	0.0000
Never Retrain ETS	0	1944.71	42413385.75	2247.71	0.00	1944.71	1944.71	0.00	0.0000
Always Retrain ARIMA	100	186.52	432435.96	235.94	1.90	186.52	188.42	**115.6**	0.0064
Always Retrain ETS	100	163.73	331660.73	214.54	1.82	163.73	**165.55**	111.0	0.0061
AWS Retrain ARIMA	60.3	210.15	544797.63	265.39	1.15	210.15	211.30	69.6	0.0038
AWS Retrain ETS	54.7	218.43	670828.37	295.75	1.00	218.43	219.43	60.7	0.0033
TS ARIMA	29.8	357.74	2123711.33	507.48	0.57	357.74	358.30	34.4	0.0019
TS ETS	29.4	805.20	11815265.78	1161.47	0.54	805.20	805.74	32.6	0.0018
TS Adapted ARIMA	65.7	209.66	562254.38	274.73	1.25	209.66	210.91	76.0	0.0042
TS Adapted ETS	60.0	333.53	2698491.76	564.64	1.09	333.53	334.62	66.6	0.0037

Table 5.5: Results of 10 Hourly Time Series, all Costs in €, TS = Tracking Signal, Total Cost = Cost Training + Cost Error

6 DISCUSSION OF RESULTS REGARDING HYPOTHESIS

One of the main hypotheses is that the cost-benefit relation of the tracking signal monitoring is better then the AWS monitoring method which is based on the in-sample error measures (MAE, MSE, RMSE) since retrainings at almost every data point are expected with the AWS retraining method. This hypothesis can only conditionally be confirmed. The AWS monitoring results in about 60 retrainings over both datasets and models. Therefore, the expectations of a retraining at almost every data point can not fully be validated. The in-sample error measures are a high standard for further predictions but in many cases, the error of the forecasts stays below the threshold. This is observed mostly when the trend and seasonality of the time series is stable and especially with observations that show almost the same value as the ones prior to it. The tracking signal monitoring with the threshold of 1.2 is able to reduce the amount of retraining signals to about 26. However the resulting higher error measures might come at a high costs. We set the costs for an increased error at € 0.01 $*$ MAE $*$ monitoring horizon which leads to higher total costs with the tracking signal monitoring then the AWS monitoring. With a lower cost factor for reduced performance this relation will change. This cost factor is highly dependent on the usecase and specific requirements. With millions of parallel time series in production, maybe a increased error is accepted when the training costs for a single time series can be reduced.

The other factor influencing this relation is the cost of the training. With both model types examined here, relatively fast training times resulting in low training costs are experienced. A single retraining with the ETS model takes a few seconds and for the ARIMA model up to a minute retraining time is required. The major part of the cost calculation for the retraining consists of the startup time of about 5 minutes required for the computing instance to be prepared before the actual calculation is executed. Since this startup time is added at every retraining, the computing time itself can almost be neglected. We still see the heightened costs for the ARIMA model over the ETS model. With other model types, especially with ML models like neural networks or hybrid models, the computational requirements are expected to be higher and thus much more expensive leading to the desire to keep the retraining counts at a minimum. Therefore, the cost-benefit relation of the tracking signal can only surpass the default AWS model quality monitoring when either the training costs are high or a reduced model performance can be accepted in favor of lower retraining costs.

The adapted tracking signal which is designated to result in a comparable amount of retrainings to the AWS monitoring method resulted in a threshold of 0.4 over both datasets and gives comparable but slightly higher retrain-

ing counts. At this level, the adapted tracking signal monitoring results in an error increase of about 13 % compared to the AWS monitoring for the daily dataset and about 25% for the hourly dataset where the latter needs to be interpreted with the knowledge of the generally bad fit of the ETS model in this dataset with a forecasting horizon greater then 1. This poor performance of the ETS model with the hourly data in combination with the adapted tracking signal is also visible with the AWS monitoring but on a lower scale, suggesting that the retraining timings are selected better than with the adapted tracking signal monitoring. To give a generally valid assessment of the performance of the tracking signal monitoring against the default AWS model quality monitoring, adjusted AWS monitoring thresholds are required to be tested against multiple tracking signal thresholds. Here, only one direct comparison is evaluated that indicates the superior performance of the AWS monitoring on that level.

The tracking signal monitoring can be seen as a better fit in an MLOps environment due to the more generic approach of adjusting the desired retraining amount. Over both datasets and models, the tracking signal threshold of 1.2 leads to 25-30 retrainings and the threshold of 0.4 results in 60-65 retrainings. This finding shows that certain threshold values result in related retraining amounts so no further knowledge of the data or the in-sample error metrics are required when aiming for a specific level of retrainings. This finding should be validated in further experiments. The AWS method results in 54-60 retrainings but to adjust this level, more knowledge about the data and the in-sample error metrics are required to enable the manual adjustment.

The monitoring framework also gives an overview about the costs generated by the training on cloud computing instances, the business costs generated by a decreased model performance and the environmental costs generated by the computing instances. We have no comparable costs for a degraded model performance at hand so the set value is considered high. With this assumption, the hypothesis of the always retrain method as economically cheapest approach for the models tested can be confirmed but ecologically the highest costs are generated due to the maximum retraining count. The ecological costs are in general at a low level with one time series over a monitoring horizon of 100 and potentially 100 retrainings only producing up to about 120g of CO_2. This amount is the CO_2 output equivalent of about 2 liters of water heated in a electric kettle [Sch15] and can be translated to below € 0.01 when using the price of CO_2 emission certificates.

7 CONCLUSION AND FURTHER RESEARCH

The goal of this research is to evaluate the performance and usability of a tracking signal in a time series forecasting MLOps environment including economical and ecological costs. As a benchmark, an AWS MLOps architecture and its default model monitoring is introduced beside the two extreme methods of always and never retrain the models. The time series forecasting models are limited to ARIMA and ETS models. With the results at hand, it is not possible to give a global recommendation to any of the methods tested. When retrainings are expensive, which is expected especially for more complex ML or hybrid models which should be evaluated in further research, the tracking signal monitoring might be the cheapest model monitoring solution. Another benefit of the tracking signal is the ability to easily adjust the monitoring threshold to a level of an expected retraining outcome. With the adapted tested threshold value of 0.4 which functions as direct comparison against the default AWS monitoring, the performance falls behind the AWS monitoring and can thus not be recommended. Further research is required with additional threshold values for the tracking signal and also adjusted values for the error metrics of the AWS method in combination with additional data to come to a final conclusion.

We only tested one tracking signal with a specific parameter set. Different parameters like different smoothing factors or a changed calculation of the tracking signal are further possible research options to generate additional insights regarding the performance of the tracking signal monitoring against the AWS model monitoring approach with in-sample error metrics.

The largest part of training costs experienced here is based on the startup time of the AWS EC2 instances prior to the actual computing time. Besides the costs that can be reduced by a different scheduling of retrainings, the organisation of the MLOps framework should be evaluated as well. When multiple time series are updated at the same time, only one or a few instances should be started and a sequential computation of the retraining should be implemented to further cut costs. This is recommended especially with models of low computational retraining requirements.

The ecological costs evaluated in this research are on a low level, however when utilizing such an MLOps environment as described here with AWS, potentially millions of time series and models can be managed so the responsibility of reducing costs should also apply to the reduction of the ecological footprint.

The examined monitoring in this research is limited to the model quality monitoring with ground truth data available on a regular basis. In a different scenario, monitoring approaches that do not require further ground truth data to create alerts like described in chapter 2.3 may be used additionally.

The influence between those different approaches on the model performance and the costs generated can be evaluated in further research.

The evaluation of the best monitoring methods to reduce required retrainings in a time series forecasting problem is a topic which is not considered on a broad scientific scope. With this research and the standard models and datasets used, a benchmark is given for further research, especially in combination with MLOps frameworks.

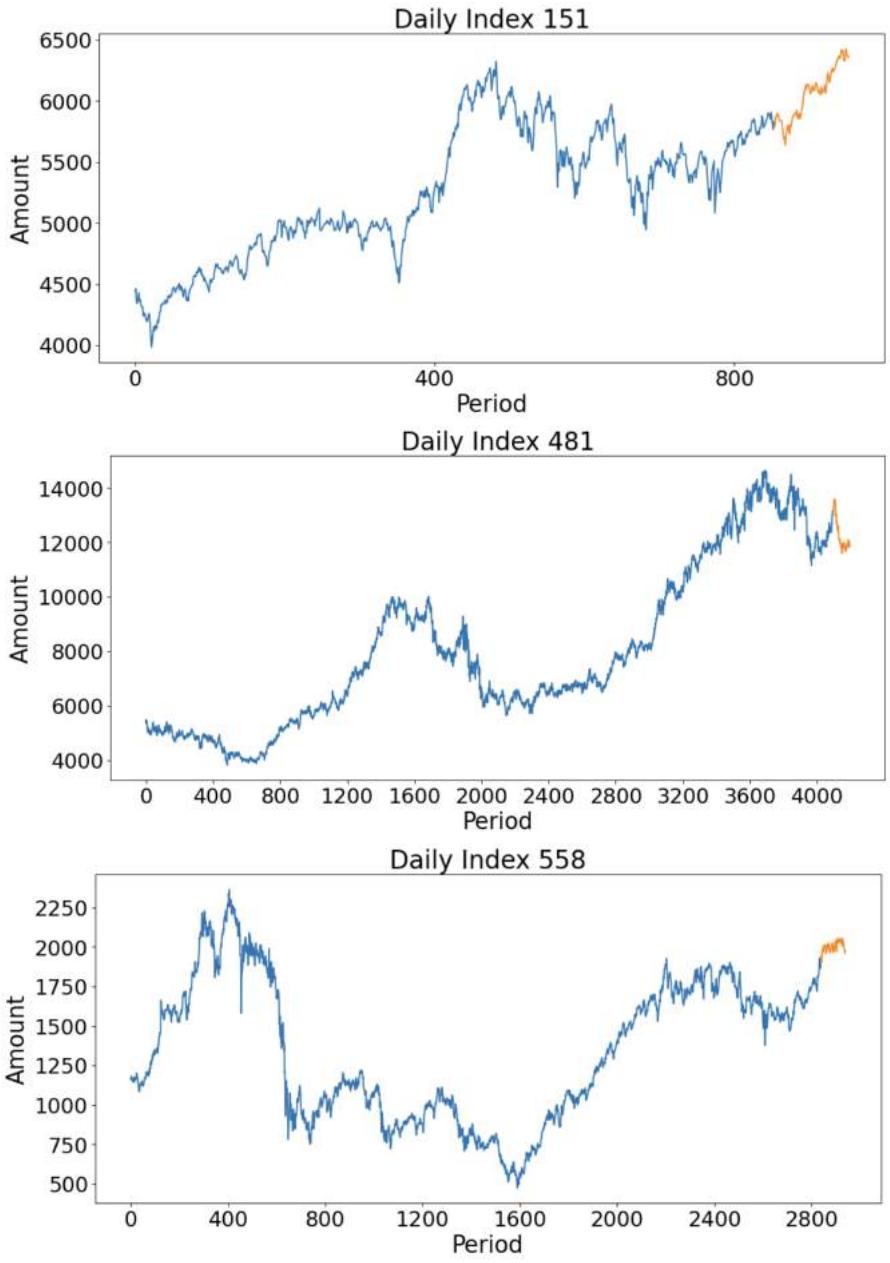
Part II

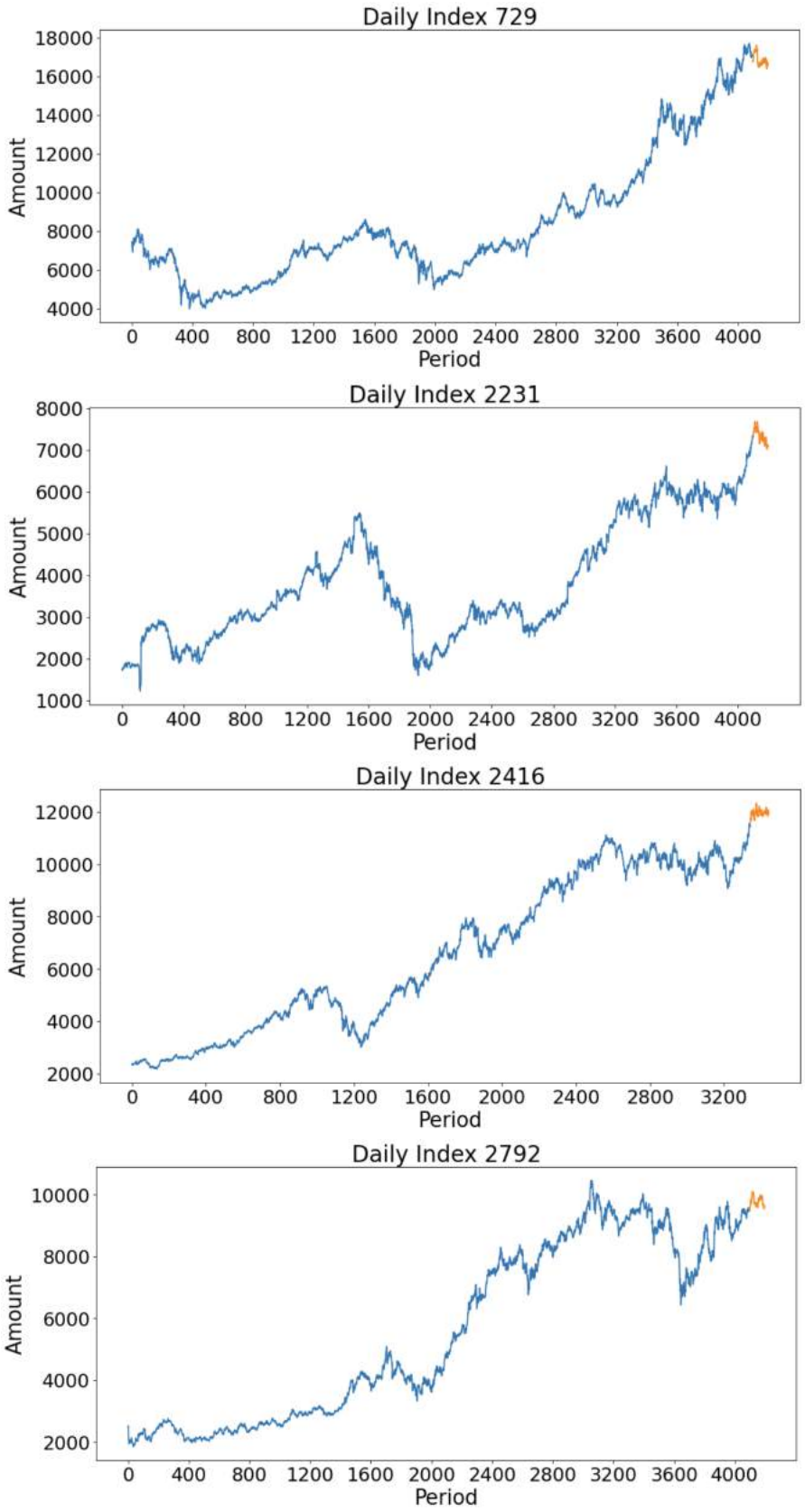
APPENDIX

A

ADDITIONAL FIGURES AND TABLES

A.1 10 SAMPLED TIME SERIES OF DAILY M4 DATASET




Daily Index 729
Amount
Period
Daily Index 2231
Amount
Period
Daily Index 2416
Amount
Period
Daily Index 2792
Amount
Period

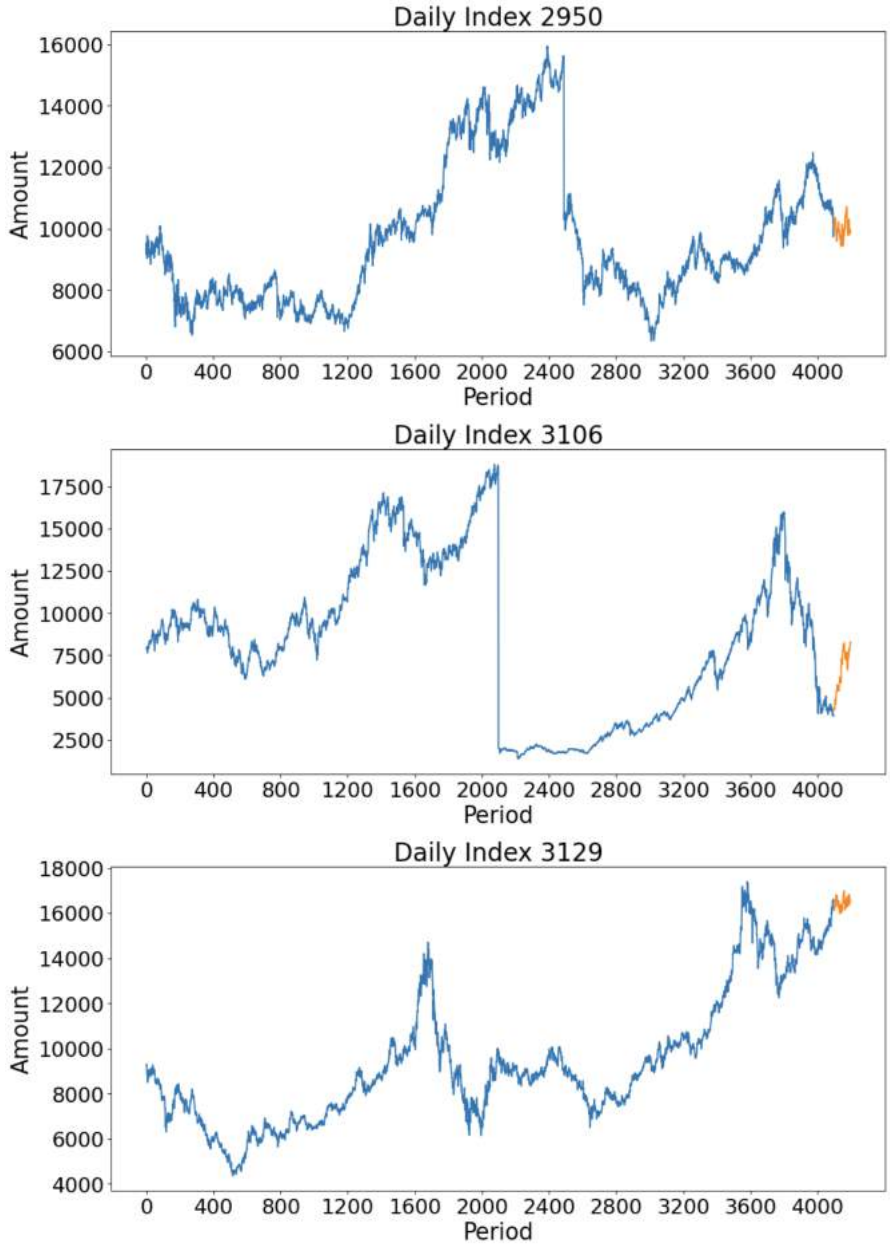
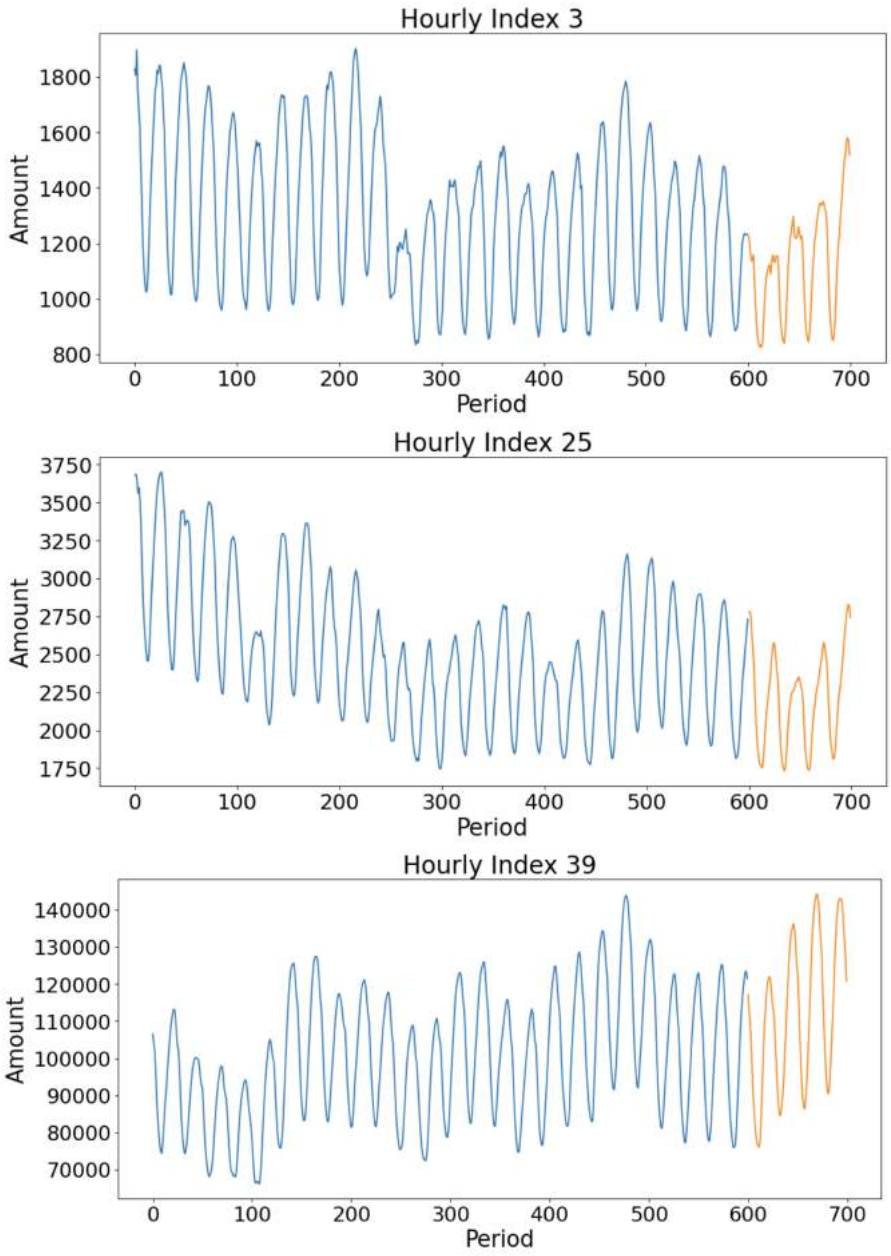
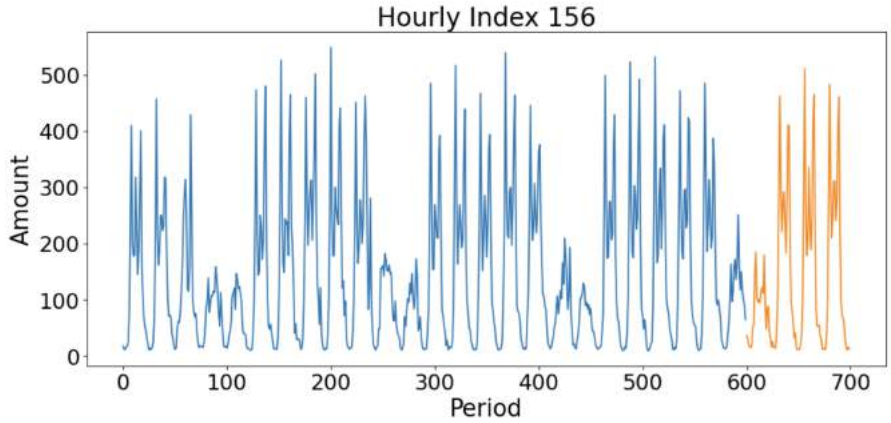
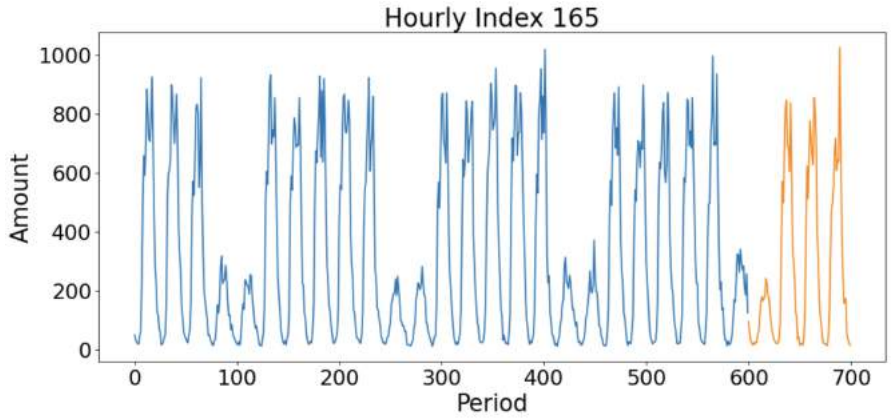


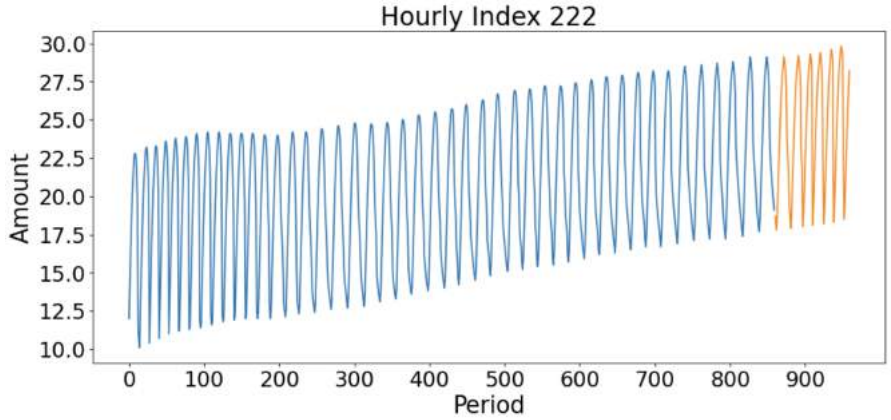
Figure A.1: 10 Random Samples of Daily M4 Dataset with Training Data and 100 Test Datapoints

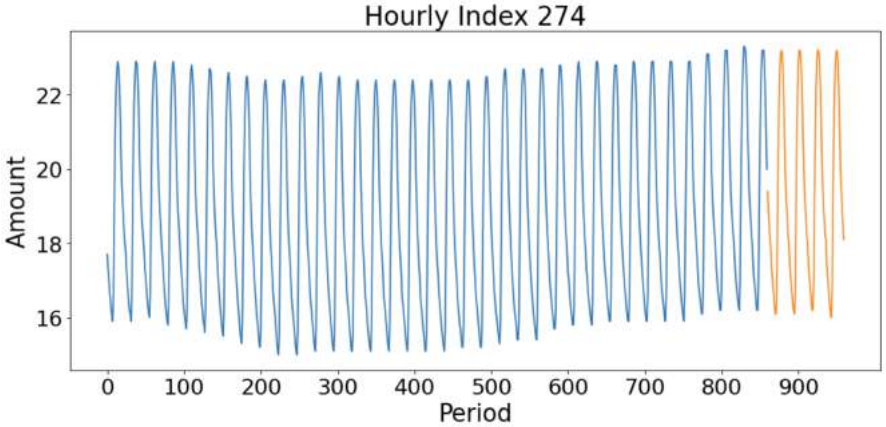
A.2 10 sampled time series of hourly m4 dataset




Hourly Index 156
Amount
Period
0
100
200
300
400
500
600
700


Hourly Index 165
Amount
Period
0
200
400
600
800
1000


Hourly Index 222
Amount
Period
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
900


Hourly Index 274
Amount
Period
16
18
20
22
900

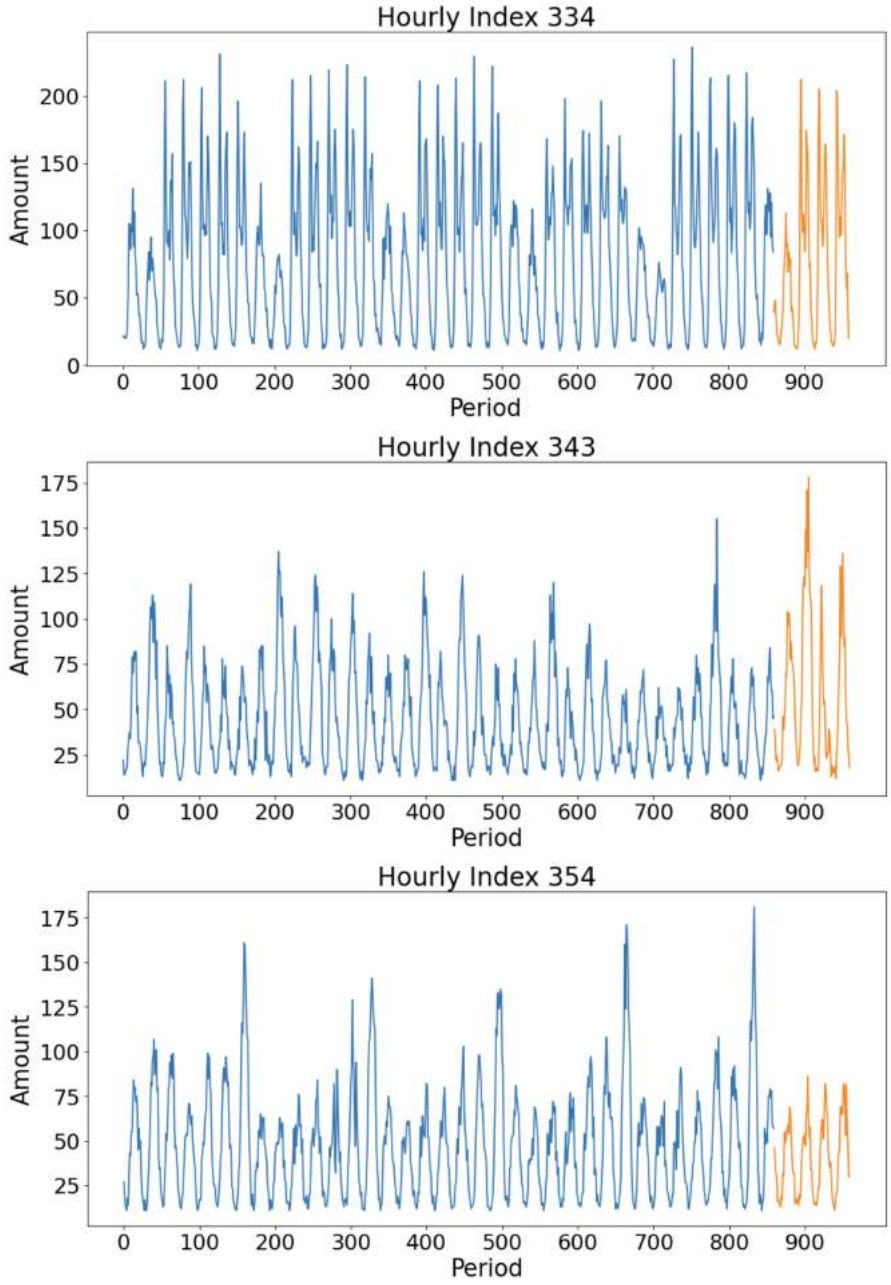


Figure A.2: 10 Random Samples of Hourly M4 Dataset with Training Data and 100 Test Datapoints

A.3 ADDITIONAL FIGURES FROM OWN EXPERIMENT

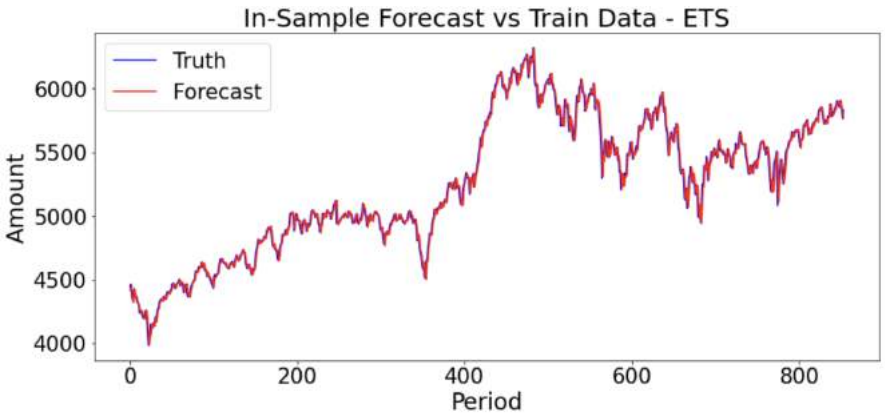


Figure A.3: In-Sample Forecasts vs Training Data for ETS Model - Daily Example

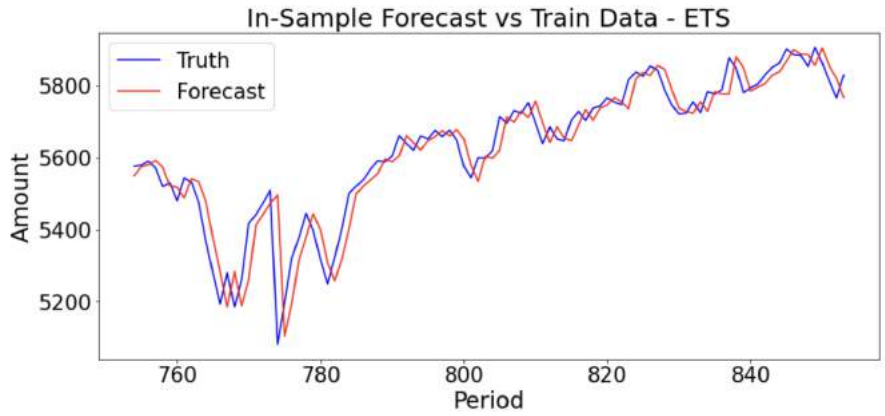


Figure A.4: In-Sample Forecasts vs Training Data for ETS Model, last 100 Observations - Daily Example

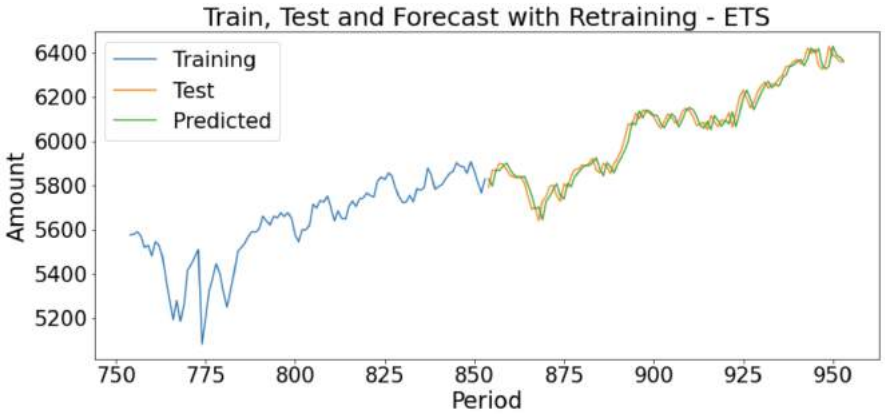


Figure A.5: Always Retrain Forecasts vs Training Data for ETS Model - Daily Example

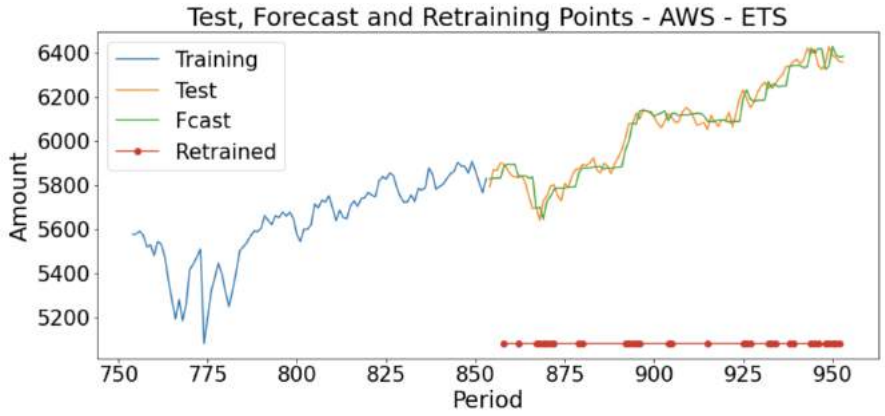


Figure A.6: AWS Retrain ETS Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Daily Example

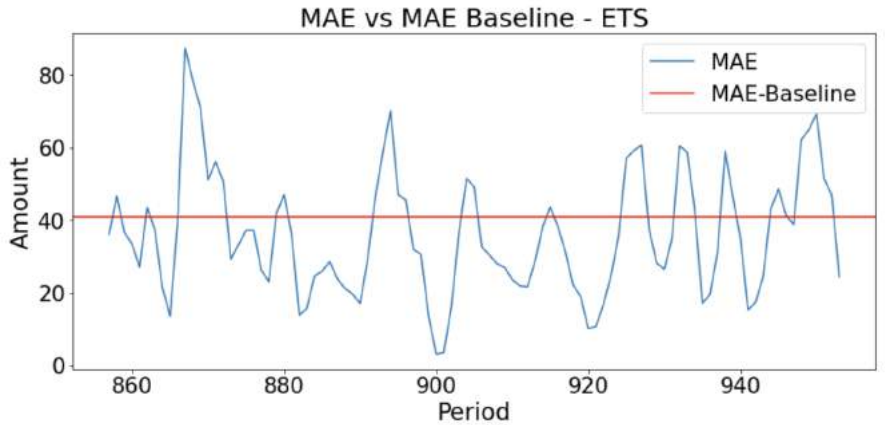


Figure A.7: MAE Over Time - ETS - Daily Example

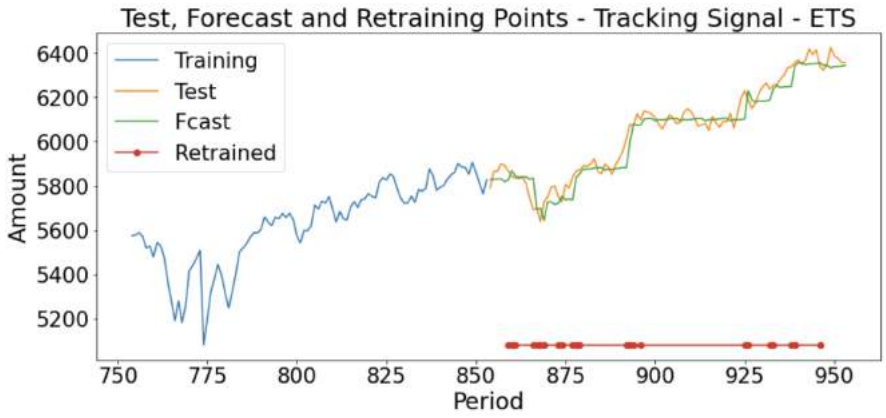


Figure A.8: Tracking Signal Monitoring ETS Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Daily Example

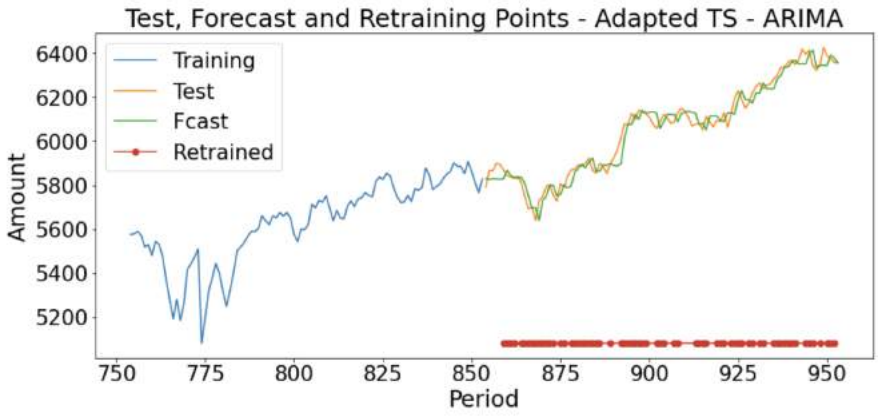


Figure A.9: Adapted Tracking Signal Monitoring ARIMA Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Daily Example

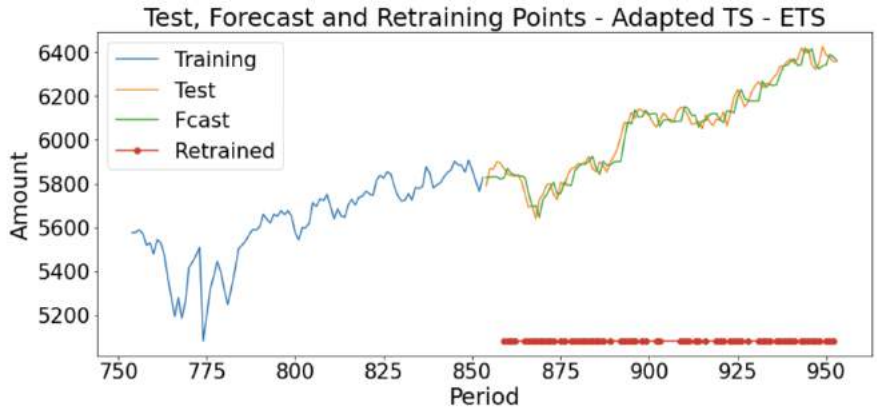


Figure A.10: Adapted Tracking Signal Monitoring ETS Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Daily Example

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=5735.550, Time=0.50 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=6776.396, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=5934.885, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=6301.494, Time=0.12 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=6774.520, Time=0.01 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=5921.813, Time=0.25 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=5769.206, Time=0.35 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=5678.474, Time=0.62 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=5748.159, Time=0.33 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=5749.713, Time=0.62 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=5680.177, Time=0.74 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=5699.763, Time=0.44 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=5734.029, Time=0.42 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=5680.656, Time=0.69 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=5676.909, Time=0.33 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=5734.410, Time=0.19 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=5747.175, Time=0.21 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=inf, Time=0.50 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=5678.625, Time=0.44 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=5768.613, Time=0.15 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=5698.428, Time=0.25 sec
 ARIMA(4,1,1)(0,0,0)[0]             : AIC=5732.957, Time=0.22 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=5679.101, Time=0.44 sec

Best model:  ARIMA(3,1,2)(0,0,0)[0]
```

Figure A.11: Iterative Auto-ARIMA Process - Hourly Example

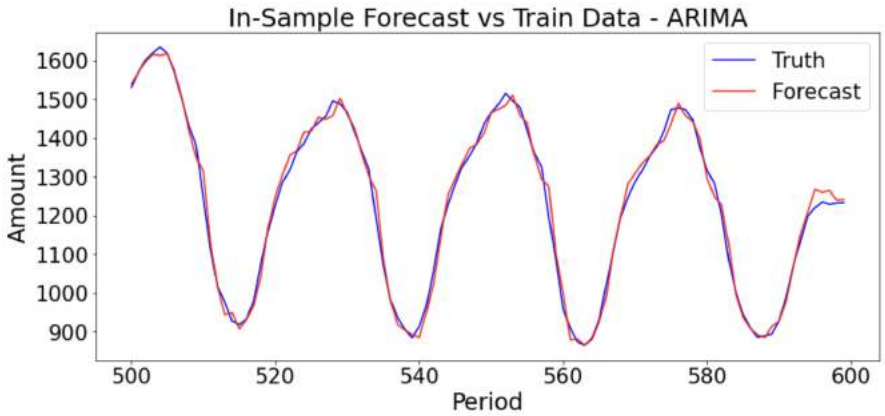


Figure A.12: In-Sample Forecasts vs Training Data for ARIMA Model - Hourly Example

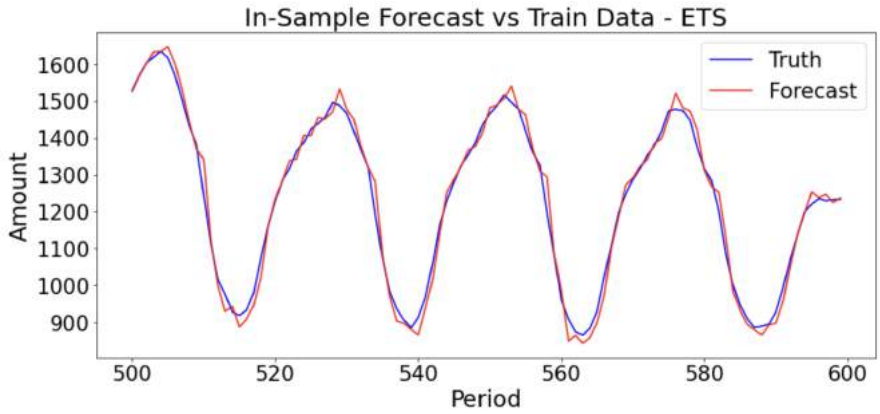


Figure A.13: In-Sample Forecasts vs Training Data for ETS Model - Hourly Example

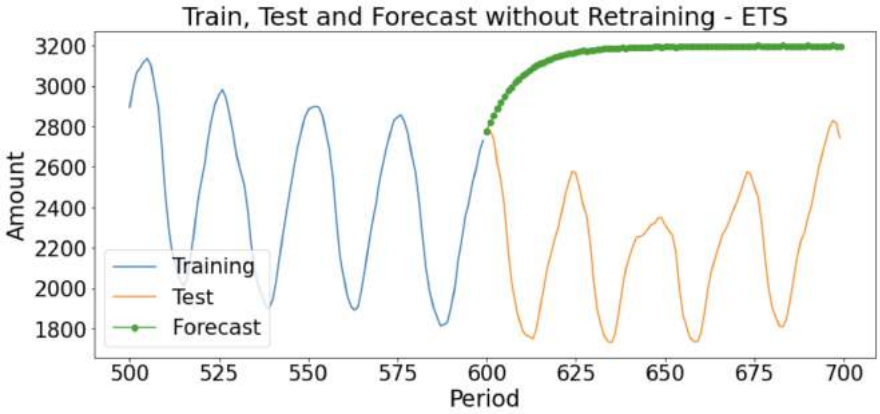


Figure A.14: Further Never Retrain Example Daily Dataset Index 25

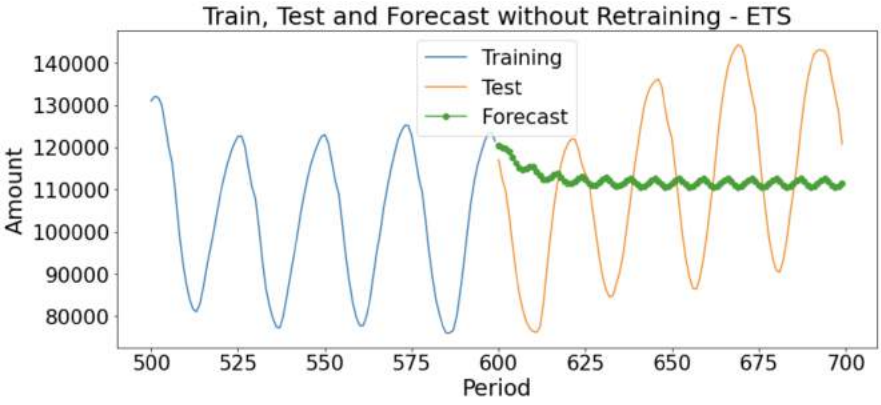


Figure A.15: Further Never Retrain Example Daily Dataset Index 39

	N Retrain	MAE	MSE	RMSE	Cost Training	Cost Error	Total Costs	Co2 [g]	Co2 [€]
In-Sample ARIMA		20.38	789.44	28.10					
In-Sample ETS		25.37	1157.59	34.02					
Never Retrain ARIMA	0	122.52	26224.56	161.94	0.00	122.52	122.52	0.00	0.0000
Never Retrain ETS	0	174.60	47867.60	218.79	0.00	174.60	174.6	0.00	0.0000
Always Retrain ARIMA	100	21.43	784.13	28.00	1.88	21.43	**23.31**	**114.48**	0.0063
Always Retrain ETS	100	24.38	1041.55	32.27	1.82	24.38	26.20	111.01	0.0061
AWS Retrain ARIMA	68	24.64	953.38	30.88	1.28	24.64	25.92	77.84	0.0043
AWS Retrain ETS	65	30.68	1502.95	38.77	1.19	30.68	31.87	72.16	0.0040
TS ARIMA	32	45.89	3833.20	61.91	0.60	45.89	46.49	36.63	0.0020
TS ETS	36	85.21	14082.41	118.67	0.66	85.21	85.87	39.97	0.0022
TS Adapted ARIMA	64	30.77	1652.65	40.65	1.22	30.77	31.99	74.25	0.0041
TS Adapted ETS	60	34.11	2163.99	46.52	1.09	34.11	35.20	66.59	0.0037

Table A.1: Results of Hourly Example Time Series, all Costs in €, TS = Tracking Signal, Total Cost = Cost Training + Cost Error

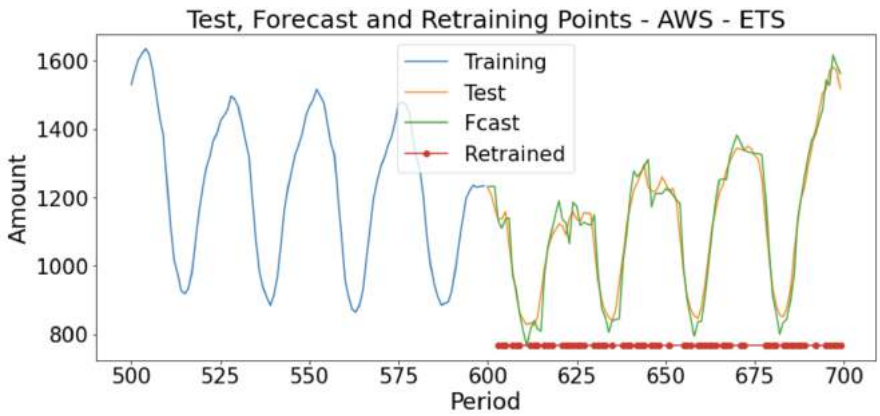


Figure A.16: AWS Retrain ETS Model, last 100 Training Observations, Forecasts, Test Data and Retrain Timings - Hourly Example

BIBLIOGRAPHY

[Agr+19] Pulkit Agrawal et al. "Data Platform for Machine Learning." In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1803–1816. ISBN: 9781450356435. DOI: 10.1145/3299869.3314050.

[AC20] Tanveer Ahmad and Huanxin Chen. "A review on machine learning forecasting growth trends and their real-time applications in different energy systems." In: *Sustainable Cities and Society* 54 (2020). DOI: https://doi.org/10.1016/j.scs.2019.102010.

[AA20] Sridhar Alla and Suman Kalyan Adari. *Beginning MLOps with MLFlow - Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*. New York: Apress, 2020. ISBN: 978-1-484-26548-2.

[Ath14] Rob J. Hyndman; George Athanasopoulos. *Forecasting: Principles and Practice*. Paperback. Otexts, 2014. ISBN: 9780987507105.

[ACBG02] Peter Auer, Nicolò Cesa-Bianchi, and Claudio Gentile. "Adaptive and Self-Confident On-Line Learning Algorithms." In: *Journal of Computer and System Sciences* 64.1 (2002), pp. 48–75. DOI: https://doi.org/10.1006/jcss.2001.1795.

[Bal+21] Raj Bala, Bob Gill, Dennis Smith, David Wright, and Kevin Ji. *Magic Quadrant for Cloud Infrastructure and Platform Services*. Last accessed 28 Mai 2022. 2021. URL: https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sbv.

[BEA21] A. Barrak, E.E. Eghan, and B. Adams. "On the Co-evolution of ML Pipelines and Source Code - Empirical Study of DVC Projects." In: *2021 IEEE Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2021, pp. 422–433. DOI: 10.1109/SANER50967.2021.00046.

[Bay+19] Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, Hui Miao, Neoklis Polyzotis, Mitchell Trott, and Martin Zinkevich. "Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform." In: *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. Santa Clara, CA: USENIX Association, 2019, pp. 51–53. ISBN: 978-1-939133-00-7.

[Bay+17] Denis Baylor et al. "TFX: A TensorFlow-Based Production-Scale Machine Learning Platform." In: *KDD '17: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1387–1395. DOI: 10.1145/3097983.3098021.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[Bis19] Ekaba Bisong. *Building Machine Learning and Deep Learning Models on Google Cloud Platform - A Comprehensive Guide for Beginners*. New York: Apress, 2019. ISBN: 978-1-484-24469-2.

[BO93] Bruce L Bowerman and Richard T O'Connell. "Forecasting and time series: An applied approach. 3rd." In: (1993).

[BJ70] G. E. P. Box and G. M. Jenkins. *Time series analysis: Forecasting and control*. San Francisco, USA: Holden Day, 1970.

[Bri+21] Julian Bright, Alessandro Cerè, Georgios Schinas, and Theiss Heilker. *Automate model retraining with Amazon SageMaker Pipelines when drift is detected*. Last accessed 08 Jun 2022. 2021. URL: https://aws.amazon.com/de/blogs/machine-learning/automate-model-retraining-with-amazon-sagemaker-pipelines-when-drift-is-detected/.

[Bro59] Robert Goodell Brown. *Statistical Forecasting for Inventory Control*. New York: McGraw-Hill, 1959.

[Bro62] Robert Goodell Brown. *Smoothing Forecasting and Prediction of Discrete Time Series*. Englewood Cliffs: Prentice-Hall, 1962.

[Bro63] Robert Grover Brown. *Smoothing, Forecasting and Prediction of Discrete Time Series* -. New York: Prentice-Hall, 1963.

[Bun22] Bundesregierung. *CO2 hat einen Preis - Anreiz für weniger CO2-Emissionen*. 2022. URL: https://www.bundesregierung.de/breg-de/themen/klimaschutz/weniger-co2-emissionen-1790134.

[Can+21] Rosa Candela, Pietro Michiardi, Maurizio Filippone, and Maria A. Zuluaga. "Model Monitoring and Dynamic Model Selection in Travel Time-Series Forecasting." In: *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track*. Ed. by Yuxiao Dong, Dunja Mladenić, and Craig Saunders. Cham: Springer International Publishing, 2021, pp. 513–529. DOI: https://doi.org/10.1007/978-3-030-67667-4_31.

[Cao+20] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. "An Overview on Edge Computing Research." In: *IEEE Access* 8 (2020), pp. 85714–85728. DOI: 10.1109/ACCESS.2020.2991734.

[CTS19] Vitor Cerqueira, Luis Torgo, and Carlos Soares. *Machine Learning vs Statistical Methods for Time Series Forecasting: Size Matters*. 2019. DOI: 10.48550/ARXIV.1909.13316.

[CML14] Min Chen, Shiwen Mao, and Yunhao Liu. "Big Data: A survey." In: *Mobile Networks and Applications* 19.2 (2014), pp. 171–209. DOI: 10.1007/s11036-013-0489-0.

[CY04] Zhuo Chen and Yuhong Yang. "Assessing forecast accuracy measures." In: *Preprint Series* 2010 (2004), pp. 2004–10.

[COH14] Yi-Ju Chiang, Yen-Chieh Ouyang, and Ching-Hsien Hsu. "An efficient green control algorithm in cloud computing for cost optimization." In: *IEEE Transactions on Cloud Computing* 3.2 (2014), pp. 145–155.

[Clo22] Google Cloud. *MLOps: Continuous delivery and automation pipelines in machine learning*. Last accessed 24 Mai 2022. 2022. URL: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[CGG09] Jacqueline Cohen, Samuel Garman, and Wilpen Gorr. "Empirical calibration of time series monitoring methods using receiver operating characteristic curves." In: *International Journal of Forecasting* 25.3 (2009), pp. 484–497.

[Com20] European Commission. "White Paper on Artificial Intelligence - A European approach to excellence and trust." In: *COM(2020)* 65 final (2020).

[Com21] European Commission. "Proposal for a regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts." In: *COM(2021)* 206 final (2021).

[DF13] Andrey Davydenko and Robert Fildes. "Measuring forecasting accuracy: The case of judgmental adjustments to SKU-level demand forecasts." In: *International Journal of Forecasting* 29.3 (2013), pp. 510–522. DOI: https://doi.org/10.1016/j.ijforecast.2012.09.002.

[DF16] Andrey Davydenko and Robert Fildes. "Forecast Error Measures: Critical Review and Practical Recommendations." In: 2016. DOI: 10.13140/RG.2.1.4539.5281.

[DH06] Jan G. De Gooijer and Rob J. Hyndman. "25 years of time series forecasting." In: *International Journal of Forecasting* 22.3 (2006), pp. 443–473. DOI: https://doi.org/10.1016/j.ijforecast.2006.01.001.

[Den+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[DJI04] Yan Deng, Majid Jaraiedi, and Wafik H Iskander. "Tracking signal test to monitor an intelligent time series forecasting model." In: *Intelligent Manufacturing*. Vol. 5263. SPIE. 2004, pp. 149–160.

[For22] International Institute of Forecasters. *Announcing the M6 forecasting competition*. 2022. URL: https://forecasters.org/blog/2022/01/19/announcing-the-m6-forecasting-competition/.

[Fou22] National Science Foundation. *ABOUT ICE CORES*. Last accessed 19 Jul 2022. 2022. URL: https://icecores.org/about-ice-cores.

[GG16] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[GM05] Wilpen L Gorr and Shannon A McKay. "Application of tracking signals to detect time series pattern changes in crime mapping systems." In: *Geographic information systems and crime analysis*. IGI Global, 2005, pp. 171–182.

[Gre+17] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber. "The Sacred Infrastructure for Computational Research." In: *Proceedings of the 16th Python in Science Conference*. 2017, pp. 49–56. DOI: 10.25080/shinma-7f4c6e7-008.

[HBB22] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. "Global models for time series forecasting: A Simulation study." In: *Pattern Recognition* 124 (2022), p. 108441. DOI: https://doi.org/10.1016/j.patcog.2021.108441.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[HLV16] Mark Holmstrom, Dylan Liu, and Christopher Vo. "Machine learning applied to weather forecasting." In: *Meteorol. Appl* 10 (2016), pp. 1–5.

[Hol57] Charles C. Holt. *Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages* -. New York: Defense Technical Information Center, 1957.

[HA18] R.J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. 2nd ed. Last Accessed 02 June 2022. Melbourne, Australia: OTexts, 2018. URL: https://otexts.com/fpp2/.

[Hyn20] Rob J. Hyndman. "A brief history of forecasting competitions." In: *International Journal of Forecasting* 36.1 (2020), pp. 7–14. DOI: https://doi.org/10.1016/j.ijforecast.2019.03.015.

[JM17] Garima Jain and Bhawna Mallick. *A Study of Time Series Models ARIMA and ETS*. 2017. DOI: http://dx.doi.org/10.2139/ssrn.2898968.

[Jan93] J-SR Jang. "ANFIS: adaptive-network-based fuzzy inference system." In: *IEEE transactions on systems, man, and cybernetics* 23.3 (1993), pp. 665–685.

[Jou+17] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.

[KT03] Lutz Kilian and Mark P. Taylor. "Why is it so difficult to beat the random walk forecast of exchange rates?" In: *Journal of International Economics* 60.1 (2003), pp. 85–107. DOI: https://doi.org/10.1016/S0022-1996(02)00060-0.

[Kom22] Europäische Kommission. *EU-Emissionshandelssystem (EU-EHS)*. Last accessed 20 Jul 2022. 2022. URL: https://ec.europa.eu/clima/eu-action/eu-emissions-trading-system-eu-ets_de.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[LeC+99] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. "Object recognition with gradient-based learning." In: *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[LS09] Thomas H. Lotze and Galit Shmueli. "How does improved forecasting benefit detection? An application to biosurveillance." In: *International Journal of Forecasting* 25.3 (2009), pp. 467–483. DOI: https://doi.org/10.1016/j.ijforecast.2008.11.012.

[Mak22] Spyros Makridakis. *The M6 Financial Forecasting Competition*. 2022. URL: https://m6competition.com/.

[Mak+82] Spyros Makridakis, Allan Andersen, Robert Carbone, Robert Fildes, Michele Hibon, Rudolf Lewandowski, Joseph Newton, Emanuel Parzen, and Robert Winkler. "The accuracy of extrapolation (time series) methods: Results of a forecasting competition." In: *Journal of forecasting* 1.2 (1982), pp. 111–153. DOI: 10.1002/for.3980010202.

[MH00] Spyros Makridakis and Michèle Hibon. "The M3-Competition: results, conclusions and implications." In: *International Journal of Forecasting* 16.4 (2000), pp. 451–476. DOI: https://doi.org/10.1016/S0169-2070(00)00057-1.

[MSA18a] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "Statistical and Machine Learning forecasting methods: Concerns and ways forward." In: *PLoS ONE* 13.3 (2018). DOI: https://doi.org/10.1371/journal.pone.0194889.

[MSA18b] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "The M4 Competition: Results, findings, conclusion and way forward." In: *International Journal of Forecasting* 34.4 (2018), pp. 802–808. DOI: https://doi.org/10.1016/j.ijforecast.2018.06.001.

[MSA20] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "The M4 Competition: 100,000 time series and 61 forecasting methods." In: *International Journal of Forecasting* 36.1 (2020). M4 Competition, pp. 54–74. DOI: https://doi.org/10.1016/j.ijforecast.2019.04.014.

[MSA22] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "M5 accuracy competition: Results, findings, and conclusions." In: *International Journal of Forecasting* (2022). DOI: https://doi.org/10.1016/j.ijforecast.2021.11.013.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008. URL: http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.

[McC88] John O. McClain. "Dominant tracking signals." In: *International Journal of Forecasting* 4.4 (1988), pp. 563–572. DOI: https://doi.org/10.1016/0169-2070(88)90133-1.

[Mon07] Douglas C Montgomery. *Introduction to statistical quality control*. John Wiley & Sons, 2007.

[MJK15] Douglas C. Montgomery, Cheryl L. Jennings, and Murat Kulahci. *Introduction to Time Series Analysis and Forecasting*. New York: John Wiley & Sons, 2015. ISBN: 978-1-118-74511-3.

[Nay+18] Janmenjoy Nayak, Bighnaraj Naik, AK Jena, Rabindra K Barik, and Himansu Das. "Nature inspired optimizations in cloud computing: applications and challenges." In: *Cloud computing for optimization: Foundations, applications, and challenges* (2018), pp. 1–26.

[O'M+20] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. "Deep Learning vs. Traditional Computer Vision." In: *Advances in Computer Vision*. Ed. by Kohei Arai and Supriya Kapoor. Cham: Springer International Publishing, 2020, pp. 128–144. ISBN: 978-3-030-17795-9.

[Oye+19] Jelili Oyelade, Itunuoluwa Isewon, Olufunke Oladipupo, Onyeka Emebo, Zacchaeus Omogbadegun, Olufemi Aromolaran, Efosa Uwoghiren, Damilare Olaniyan, and Obembe Olawole. "Data Clustering: Algorithms and Its Applications." In: *2019 19th International Conference on Computational Science and Its Applications (ICCSA)*. 2019, pp. 71–81. DOI: 10.1109/ICCSA.2019.000-1.

[PIP16] Daniel Pop, Gabriel Iuhasz, and Dana Petcu. "Distributed Platforms and Cloud Services: Enabling Machine Learning for Big Data." In: *Data Science and Big Data Computing: Frameworks and Methodologies*. Ed. by Zaigham Mahmood. Cham: Springer International Publishing, 2016, pp. 139–159. ISBN: 978-3-319-31861-5.

[Qin20] Tao Qin. *Dual Learning*. Singapore: Springer Nature, 2020. ISBN: 978-9-811-58884-6.

[Ré+19] Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. *Overton: A Data System for Monitoring and Improving Machine-Learned Products*. 2019. DOI: 10.48550/ARXIV.1909.05372.

[Sch15] Caroline Schmitt. *A day in CO2e emissions*. 2015. URL: https://www.dw.com/en/a-day-in-co2e-emissions/a-18892482.

[Scu+15] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. "Hidden Technical Debt in Machine Learning Systems." In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf.

[Ser22a] Amazon Web Services. *AWS CloudFormation - Speed up cloud provisioning with infrastructure as code*. Last accessed 09 Jun 2022. 2022. URL: https://aws.amazon.com/cloudformation/.

[Ser22b] Amazon Web Services. *AWS Lambda - Run code without thinking about servers or clusters*. Last accessed 09 Jun 2022. 2022. URL: https://aws.amazon.com/lambda/.

[Ser22c] Amazon Web Services. *AWS Service Catalog - Create, organize, and govern your curated catalog of AWS products*. Last accessed 09 Jun 2022. 2022. URL: https://aws.amazon.com/servicecatalog.

[Ser22d] Amazon Web Services. *Amazon CloudWatch - Observability of your AWS resources and applications on AWS and on-premises*. Last accessed 09 Jun 2022. 2022. URL: https://aws.amazon.com/cloudwatch/.

[Ser22e] Amazon Web Services. *Amazon S3 - Object storage built to retrieve any amount of data from anywhere*. Last accessed 09 Jun 2022. 2022. URL: https://aws.amazon.com/s3/.

[Ser22f] Amazon Web Services. *Create a Model Quality Baseline*. Last accessed 19 Jul 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-model-quality-baseline.html.

[Ser22g] Amazon Web Services. *Model Quality Metrics*. Last accessed 07 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-model-quality-metrics.html.

[Ser22h] Amazon Web Services. *Monitor Bias Drift for Models in Production*. Last accessed 07 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/clarify-model-monitor-bias-drift.html.

[Ser22i] Amazon Web Services. *Monitor Feature Attribution Drift for Models in Production*. Last accessed 07 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/clarify-model-monitor-feature-attribution-drift.html.

[Ser22j] Amazon Web Services. *Monitor data quality*. Last accessed 07 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-data-quality.html.

[Ser22k] Amazon Web Services. *Monitor model quality*. Last accessed 07 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-model-quality.html.

[Ser22l] Amazon Web Services. *Monitor models for data and model quality, bias, and explainability*. Last accessed 07 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html.

[Ser22m] Amazon Web Services. *Register and Deploy Models with Model Registry*. Last accessed 09 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html.

[Ser22n] Amazon Web Services. *What Is Amazon SageMaker?* Last accessed 07 Jun 2022. 2022. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html.

[SABZ17] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices." In: *IEEE Access* 5 (2017), pp. 3909–3943. DOI: 10.1109/ACCESS.2017.2685629.

[SJ16] Galit Shmueli and Kenneth C. Lichtendahl Jr. *Practical Time Series Forecasting with R - A Hands-On Guide [2nd Edition]*. Green Cove Springs, Florida, USA: Axelrod Schnall Publishers, 2016. ISBN: 0997847913.

[Spi+20] Evangelos Spiliotis, Andreas Kouloumos, Vassilios Assimakopoulos, and Spyros Makridakis. "Are forecasting competitions data representative of the reality?" In: *International Journal of Forecasting* 36.1 (2020), pp. 37–53. DOI: https://doi.org/10.1016/j.ijforecast.2018.12.007.

[Sta22] Statista. *Umsatz mit Cloud Computing weltweit von 2010 bis 2021 und Prognose bis 2023*. Last accessed 29 Mai 2022. 2022. URL: https://de.statista.com/statistik/daten/studie/195760/umfrage/umsatz-mit-cloud-computing-weltweit/.

[SCH20] R Sujath, Jyotir Moy Chatterjee, and Aboul Ella Hassanien. "A machine learning forecasting model for COVID-19 pandemic in India." In: *Stochastic Environmental Research and Risk Assessment* 34.7 (2020), pp. 959–972.

[Sun20] Ali Sunyaev. "Cloud Computing." In: *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*. Cham: Springer International Publishing, 2020, pp. 195–236. ISBN: 978-3-030-34957-8.

[Swe88] John A Swets. "Measuring the accuracy of diagnostic systems." In: *Science* 240.4857 (1988), pp. 1285–1293.

[Sze+17] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang. "Hardware for machine learning: Challenges and opportunities." In: *2017 IEEE Custom Integrated Circuits Conference (CICC)*. 2017, pp. 1–8. DOI: 10.1109/CICC.2017.7993626.

[Tre+20] Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. *Introducing MLOps - How to Scale Machine Learning in the Enterprise*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2020. ISBN: 978-1-098-11644-6.

[Tri64] D. W. Trigg. "Monitoring a Forecasting System." In: *Operational Research Society* 15.3 (1964), pp. 241–247. DOI: https://doi.org/10.2307/3007215.

[VM21] Andrés Varón Maya. *The state of MLOps*. Last accessed 24 Mai 2022. 2021. URL: https://repositorio.uniandes.edu.co/bitstream/handle/1992/51495/23642.pdf?sequence=1&isAllowed=y.

[Ver20] Nitin Verma. *Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline*. Last accessed 09 Jun 2022. 2020. URL: https://aws.amazon.com/blogs/devops/complete-ci-cd-with-aws-codecommit-aws-codebuild-aws-codedeploy-and-aws-codepipeline/.

[Wei13] William W.S. Wei. "Time Series Analysis." In: *The Oxford Handbook of Quantitative Methods in Psychology* 2 (2013). DOI: 10.1093/oxfordhb/9780199934898.013.0022.

[Wei18] Andreas S. Weigend. *Time Series Prediction - Forecasting The Future And Understanding The Past*. New York: Routledge, 2018. ISBN: 978-0-429-96119-9.

[WR06] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[Win60] Peter R. Winters. "Forecasting sales by exponentially weighted moving averages." In: *Management Science* 6 (1960), pp. 324–342. URL: hhttp://www.jstor.org/stable/2627346.

[Yah+21] Hazha Saeed Yahia, SR Zeebaree, MA Sadeeq, NO Salim, Shakir Fattah Kak, AZ Adel, Azar Abid Salih, and Helat Ahmed Hussein. "Comprehensive survey for cloud computing based nature-inspired algorithms optimization scheduling." In: *Asian Journal of Research in Computer Science* 8.2 (2021), pp. 1–16.

[Yul27] George Udny Yule. "On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers." In: *Philosophical Transactions of the Royal Society A* 226 (1927), pp. 267–298.

[Zha+19] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. "Deep Learning Based Recommender System: A Survey and New Perspectives." In: *ACM Comput. Surv.* 52.1 (2019). DOI: 10.1145/3285029.

[ZYD20] Yue Zhou, Yue Yu, and Bo Ding. "Towards MLOps: A Case Study of ML Pipeline Platform." In: *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. 2020, pp. 494–500. DOI: 10.1109/ICAICE51518.2020.00102.

[ZBCS16] Liming Zhu, Len Bass, and George Champlin-Scharff. "DevOps and Its Practices." In: *IEEE Software* 33.3 (2016), pp. 32–34. DOI: 10.1109/MS.2016.81.