Scheduling Practices for MLOps

Table of Contents

Introduction

In the previous chapters, we have explored the essential steps in setting up MLOps resources and maintaining Azure Resource Manager (ARM) templates. These foundational chapters have equipped you with the knowledge needed to establish the infrastructure and practices required for MLOps. In this section, we delve into the core concept of MLOps and how it transforms the machine learning workflow. MLOps brings agility and automation to the ML lifecycle, ensuring that models are developed, deployed, and monitored efficiently. It is the culmination of best practices, tools, and processes that empower data scientists and engineers to collaborate effectively and deliver robust, scalable, and maintainable ML solutions.

The subsequent chapters will focus on specific aspects of MLOps, including scheduling practices, data pipeline management, model training, and real-time inference, providing you with a comprehensive understanding of how to operationalize and scale machine learning solutions in a production environment.

What is Scheduling?

Scheduling (here) refers to the process of timely execution of MLOps processes in the systematic and predefined order. All the internal components of the machine learning model run like input, output, parameters should be updated based on the automated scheduled mechanisms. A machine learning model run mainly consists of different pipelines for data input and processing, model run and reporting.

Pipeline scheduling is a critical aspect of modern data and workflow management systems, ensuring the efficient and timely execution of complex processes. In the context of data engineering and automation, a pipeline represents a series of interconnected tasks or steps that collectively achieve a specific objective, such as data extraction, transformation, and loading (ETL). Scheduling these pipelines involves orchestrating the execution of tasks at predefined intervals or in response to triggering events, ensuring seamless and reliable automation of workflows.

Scheduling in MLOps

In the realm of MLOps, machine learning workflows demand orchestration and automation for seamless model development, deployment, and monitoring. MLOps pipelines encompass a spectrum of tasks, including data pipelines, model training, evaluation, deployment, and ongoing monitoring. Efficiently scheduling these diverse tasks is essential for maintaining a robust MLOps infrastructure.

Key considerations for pipeline scheduling in MLOps involve,

1. Orchestrating workflows across various environments
2. Ensuring timely model retraining
3. Accommodating dynamic data sources.

4. Managing model versioning
5. Maintaining consistency between development and production environments
6. Automating the deployment of updated models.

A well-designed scheduling strategy in MLOps not only optimizes computational resources but also supports continuous integration and delivery (CI/CD) principles, fostering collaboration between data scientists, developers, and operations teams throughout the machine learning lifecycle. This documentation explores best practices tailored to the unique challenges of scheduling pipelines in the context of MLOps, offering insights into maximizing the efficiency and effectiveness of machine learning workflows.

Importance of Scheduling in MLOps

1. **Efficient Automation:** Scheduled workflows in MLOps automate tasks, from data collection to deployment, improving efficiency by minimizing manual intervention.
2. **Consistent and Cost-effective Operations:** Scheduled processes ensure consistent and reproducible ML tasks while optimizing resource allocation, reducing operational costs.
3. **Model Health and Real-time Insights:** Scheduled tasks cover model maintenance, real-time inference, and monitoring, ensuring models remain accurate, up-to-date, and available for immediate decision-making.
4. **Compliance and Scalability:** Scheduling incorporates compliance measures and assists in scaling resources, addressing regulatory requirements and handling fluctuations in workload.
5. **Holistic Workflow Management:** Scheduling enables the coordination of complex workflows, from data pipelines and historical tracking to continuous improvement, enhancing overall performance and traceability.

For, MLOps scheduling is concerned about **four** different types of scheduling. Data pipeline scheduling, Model training scheduling, model & data back-up scheduling.

1. Data pipeline scheduling
2. Model Training and Run Scheduling.
3. Model Back-Up Scheduling
4. Data Back-Up Scheduling

It is possible to have data pipeline run at different time and re-run the model at different times, independent of the data pipelines based on requirements. We can ingest the data time to time, and its not necessary that we run model all the time. Integrating new training data, can degrade the model. Refer to the link.

Data Pipeline Scheduling

Data pipelines are used in MLOps to **pull the input data, process/prepare the data** with different data validation checks applied. It requires timely processing before the model run, proper validation checks completion and publishing the data to proper location where ML model will take it as an input.

Data pipeline majorly are architected by Azure Data Factory, Azure Synapse pipelines, or Azure Fabric pipelines. These pipelines includes copy activities to get the data from different data sources/ servers/ and copy the data to

ADLS. Before processing the data it applied checks on data for Null integrity, incorrect data type, wrong / missing values in a field etc.

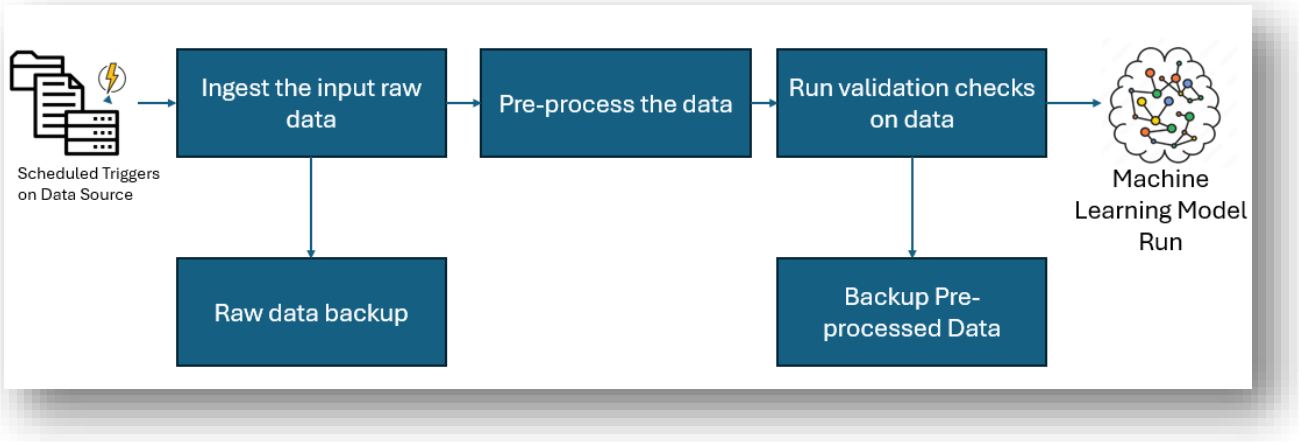
Following are some of the best data pipelines providers available:

1. Apache Airflow
    a. Apache Airflow provides advanced scheduling capabilities.
    b. It allows you to define workflows as Directed Acyclic Graphs (DAGs) and schedule them based on time, dependencies, or external triggers.
    c. It allows complex schedules, such as daily, hourly, or cron-like expressions.
    d. Airflow supports dynamic scheduling and provides a web-based UI for monitoring and managing workflows.
2. Kubeflow Pipelines
    a. Kubeflow Pipelines itself does not focus on scheduling but rather on orchestrating machine learning workflows on Kubernetes.
    b. For scheduling purposes, it can be integrated it with tools like Argo Workflows or Tekton Pipelines, which provide scheduling capabilities on Kubernetes.
3. AWS Data Pipeline
    a. AWS Data Pipeline offers scheduling capabilities for orchestrating and automating the movement and transformation of data between different AWS services.
    b. Pipelines can be defined with predefined schedules or trigger them based on events.
4. Azure Data Factory
    a. Azure Data Factory is a cloud-based data integration service with built-in scheduling capabilities.
    b. It allows to create data pipelines with activities and set up triggers to schedule the execution of these pipelines.
    c. Triggers can be time-based (e.g., hourly, daily) or event-based (e.g., data arrival).
5. Google Cloud Dataflow
    a. Google Cloud Dataflow focuses on stream and batch processing, and it doesn't have built-in scheduling features in the same way that traditional ETL tools do.
    b. However,tools such as Cloud Composer (Google Cloud's managed Apache Airflow service) or Cloud Scheduler can be used to schedule Dataflow jobs.

Things to be concerned about while working with data pipelines.

1. The frequency of your data updates. If your data is updated frequently, you will need to schedule your pipelines to run more often.
2. The complexity of your data pipelines. Complex pipelines may take longer to run, so you may need to schedule them to run at night or on weekends.
3. The availability of your compute resources. If you have limited compute resources, you may need to schedule your pipelines to run at times when there is less demand.
4. The needs of your business users. If your business users need access to the results of your data pipelines at certain times of day, you will need to schedule your pipelines to run accordingly.

Data Pipeline Flow



Data Pipeline Best Practices

1. Schedule your pipeline runs to avoid peak demand. ML pipelines can be resource-intensive, so it is important to schedule them to avoid peak demand on your compute resources. This can help to improve performance and reduce costs.
    a. Keep most of the resources in the cloud environment
    b. Make sure to keep autoscaling enabled for the resources where ever possible.
    c. Enable alerting mechanism once the pipeline runs start via Email or other means.
    d. Set up checks in initial phase of pipeline for latest availability of data and if it passes, then only let the pipeline proceed further. It will save the resources cost.
2. Use a trigger to start your pipeline runs. A trigger can be an event, such as a new file being added to a data store or a notification from an external system. Using a trigger to start your pipeline runs can help to ensure that your data is always processed in a timely manner.
    a. If the data is very frequently getting updated, setup fixed timed interval data pull to keep our system updated with the latest data.
    b. If the data is infrequent, setup the trigger for daily, monthly or weekly basis.
    c. Azure resources provides the scheduled trigger option in Data factory.
    d. If data is copied/ingested using Power Automate Flows, then multiple triggers are available based on file creation, update, deletion to trigger the flow.
    e. If there is centralized trigger scheduler available, utilize that to monitor all the triggers at once.
3. Monitor your pipeline runs to ensure that they are completing successfully. You can use monitoring tools to track the status of your pipeline runs, identify any errors, and receive alerts if there are any problems.
    a. Use centralized monitoring tools/mechanisms like Azure Monitor, as it provides monitoring of all resources including data pipelines, Azure Logic Apps along with their metric logs and alerts.
    b. Create a custom dashboard from this with required number of resources to create an ease in monitoring.
    c. If a team is not actively monitoring via monitor, then set up alert emails of pipeline stage completion or failures. Logic apps can be configured for this.
4. Have a retry mechanism in place for failed pipeline runs. In the event of a failure, it is important to have a retry mechanism in place to ensure that your pipeline run is eventually successful. This may involve retrying the run a certain number of times or retrying the run at a later time.
    a. Enable retries for every possible activities in pipelines. Three retries are recommended.
    b. Copy activity, Databricks notebook run, logging activity, should have retries enabled.

    c. A pipeline/activity should have 3 times retry enabled.
5. Use a scheduler that is designed for ML pipelines. ML pipelines can be complex and require specific resources, such as GPUs. There are a number of schedulers that are designed specifically for ML pipelines, such as Kubeflow Pipelines and MLflow.
    a. GPU requirements are majorly if you are training a deep learning models like CNN, GAN or other very complex architectures. TPUs are also available in Google environment.
    b. CUDA is provided with high compute of GPU powers by Nvidia, for parallel processing.
    c. Microsoft also provides different virtual machines backed by different processing powered GPUs. For further details of those GPUs, this documentation can be referred.
6. Check for securities while scheduling for MLOps data pipelines. Use Service Principles or Managed Identity provided by Microsoft to access a resource or to trigger a pipeline.

Following are few necessary practices for scheduling the data pipelines for MLOps.

1. Check for the data availability and schedule the trigger.
    a. If source data has a predefined data source, where file is created or deleted every day, week or other at other time frequencies. Set a trigger on data update of the data source, such that latest data can be ingested.
    b. If pipelines requirement is to run at particular time frequency, schedule a trigger for that by specifying specific date time manner.
2. Opt for scalable resources.
    a. Self Hosted Runtime:
        i. If you are converting data to parquet format, it requires Java self hosted run time in the backend to convert and compress the data. Set up auto scalability for runtime to increase number of nodes (on-prem or virtual machines) based on requirement.
        ii. Setup alerts for high usage of a resource/ or low capacity available for resource. And based on that setup resource scaling via Power Shell or other methods.
    b. Compute Resources
        i. While doing data transformation or processing using Azure Databricks or Azure ML, keep the autoscaling on for the resources. It will allow the architecture to handle multiple pipelines running simultaneously.
    c. Storage Resource
        i. Azure offers unlimited amount of storage with different pricing tiers. Configure the storage to handle multiple data pipeline simultaneously.
3. Check for reuse of the utilities.
4. Optimize the cost by choosing best architectures.
    a. Optimized different types of architectures for MLOps are provided by Microsoft and other providers.
    b. Microsoft best architectures for Databricks: Link
    c. Microsoft best architecture for Azure ML Studio: Link

Scheduling Pipelines in ADF

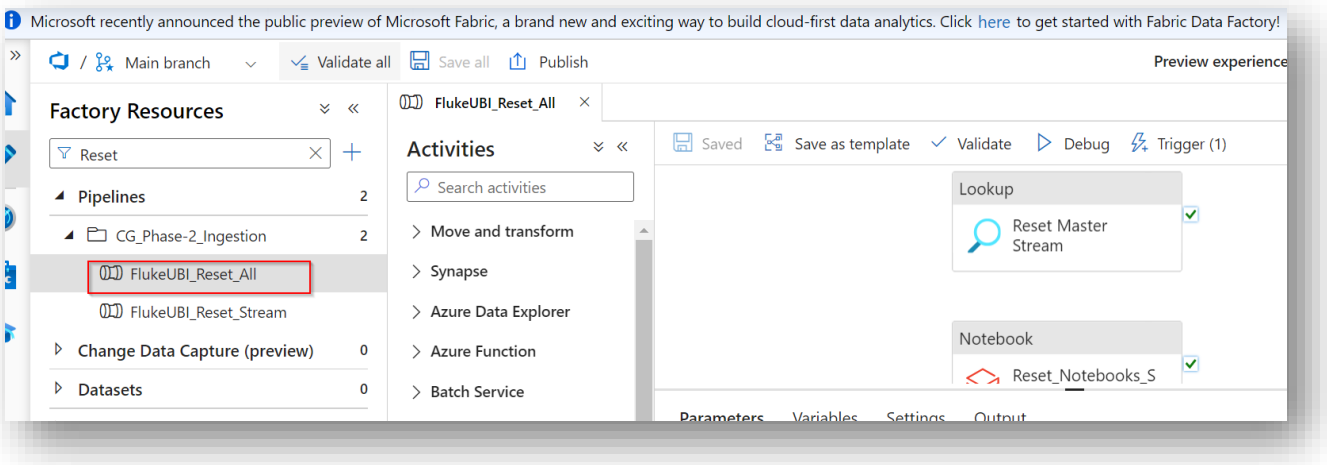
Scheduling ADF pipelines can be done using Triggers of ADF. Triggers represent a unit of processing that determines when a pipeline execution needs to be kicked off. Currently, the service supports three types of triggers:

1. Schedule trigger: A trigger that invokes a pipeline on a wall-clock schedule.
2. Tumbling window trigger: A trigger that operates on a periodic interval, while also retaining state.
    a. Tumbling windows are a series of fixed-sized, non-overlapping, and contiguous time intervals.
    b. For more information about tumbling window triggers, refer to this link.
3. Event-based trigger: A trigger that responds to an event.
    a. An event-based trigger runs pipelines in response to an event. There are two flavors of event-based triggers.
    b. Storage event trigger runs a pipeline against events happening in a Storage account, such as the arrival of a file, or the deletion of a file in Azure Blob Storage account.
4. Custom: Custom event trigger processes and handles custom articles in Event Grid

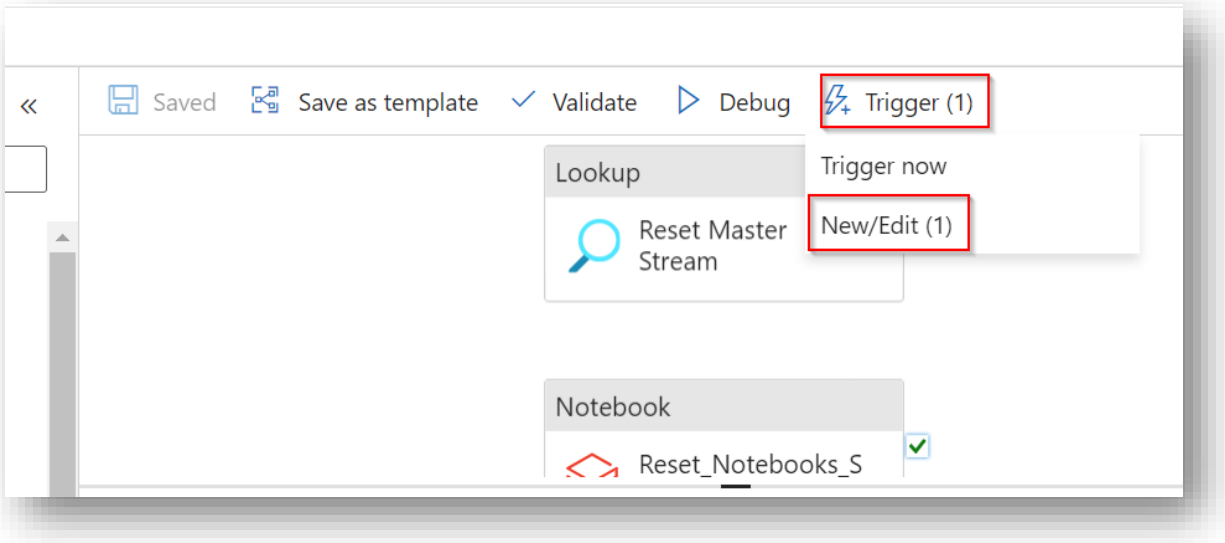
One trigger can initiate multiple pipelines and one pipeline can be associated to multiple triggers.

How to Schedule Pipeline for Trigger

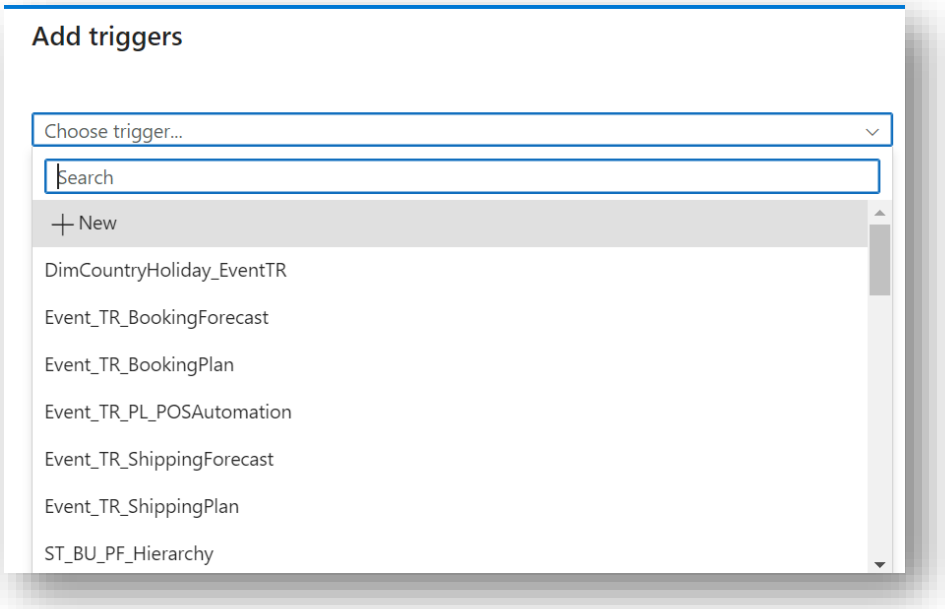
1. Open Data factory and open the pipeline which needs to be scheduled.



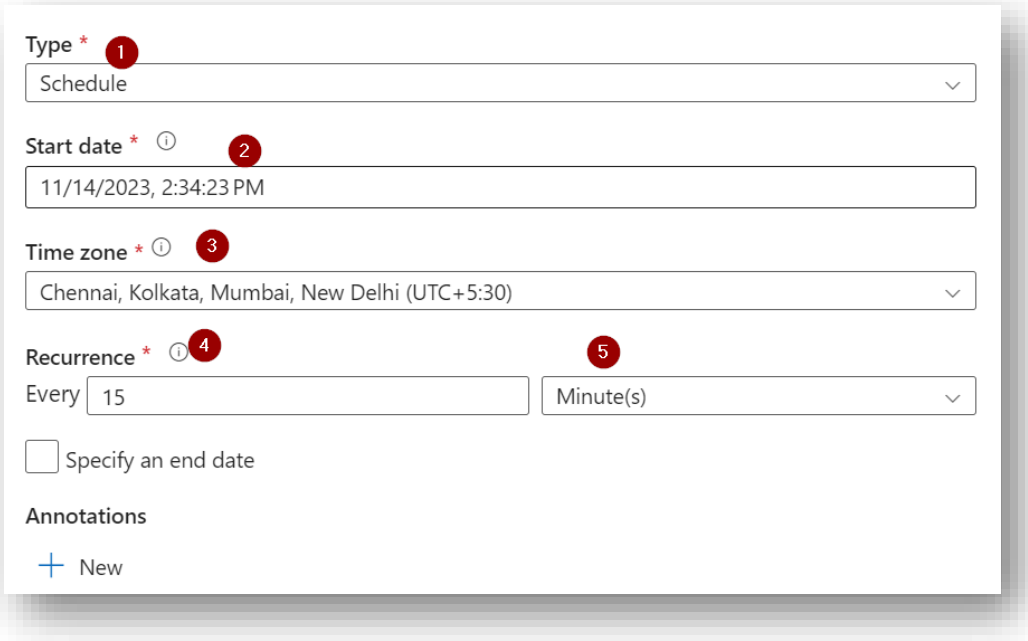
2. Click on Trigger -> New/Edit



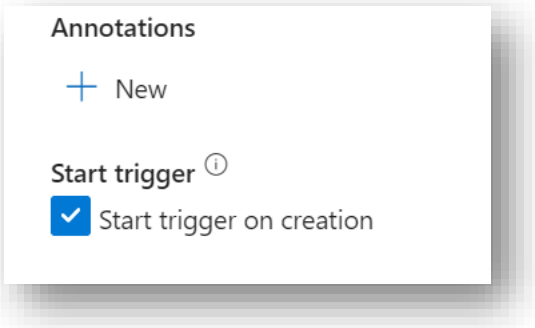
3. In the open window, click on New



4. Now select following things.
    a. Type: Scheduled/Tumbling Window/Event Based/Custom
    b. Select start datetime of the trigger.
    c. Select the followed time zone for the trigger.
    d. Add recurrence information.
        i. Recurrence can be minutes, hours, days or weeks.



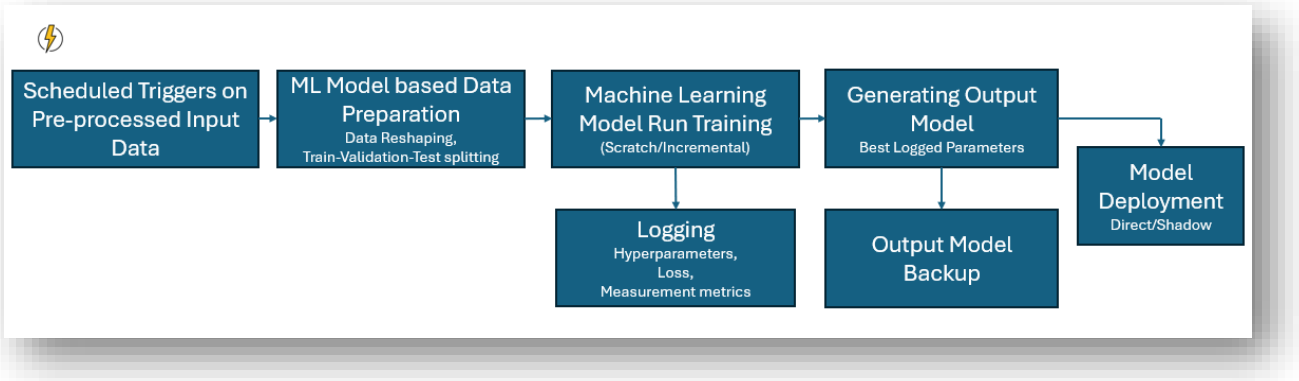
5. Select ‘Start trigger on creation’ if you want to enable the trigger and make it in active state.



Note: Once the trigger is created, pipeline needs to be published to make the trigger changes available and active.

MLOps Scheduling

Model Pipelines Flow



Model Training and Run

Machine learning model run scheduling is essential for maintaining accurate and up-to-date predictive models. By scheduling regular model runs, organizations ensure that machine learning models are retrained, evaluated, and deployed at the right intervals. This process is crucial for keeping models in sync with changing data patterns and business dynamics. Timely model runs help in maintaining predictive accuracy and ensuring that machine learning solutions provide reliable insights, which is particularly critical in decision-making processes across various domains, from financial forecasting to demand prediction.

Selection of the proper environment for MLOps architecture is key part while scheduling models and model runs. Multiple MLOps providers are available in the market currently. Below list shows some good providers

1. Azure Machine Learning
2. Amazon SageMaker
3. Atlassian BitBucket
4. Google Cloud AI Platform
5. DataRobot
6. MLFlow
7. MLJAR (Open Source and Commercial)

Triggers on the Machine Learning Model can be of different type:

1. Timely scheduled (hourly/daily/weekly) triggers
    a. Forecasting pipelines majorly are scheduled with this type of triggers.
    b. Incremental models are scheduled with this type of triggers.
2. Triggers based on new data availability.
    a. Models having cool (not frequently changing data) can have this type of triggers.

Model Training

Model training happens in three ways.

1. First time training of model
2. Re-training of model from scratch **OR** Re-training of model for new data only(incrementally)

First Training of Model

In the first-time training of a model, the process involves training a machine learning model from scratch using an initial dataset. This is typically the initial phase of model development where the algorithm learns patterns, features, and relationships within the training data. The model is initialized with random weights, and through multiple iterations (epochs), it adjusts its parameters to minimize the difference between predicted and actual outcomes. The resulting trained model is then ready for evaluation and potential deployment.

Model Re-training from Scratch

This is applicable to Forecasting models where with every new data, model is fully trained again to evaluate the possibilities and parameters between each of the months available and generate a new forecast based on that.

Apart from that in scenarios for Deep Learning models, where significant changes occur in the dataset or dataset distribution, a re-training of the model from scratch is required. This involves discarding the existing model weights and training the model anew. This process is resource-intensive and time-consuming, as the model essentially starts learning from a clean slate. Re-training from scratch is necessary when there are major shifts in the data distribution, changes in the target variable, or significant updates to the model architecture.

Model Re-training for New Data Only (Incremental Training)

In contrast to re-training from scratch, incremental re-training involves updating the existing model with new data while retaining the knowledge learned from the initial training. This approach is more efficient, especially when dealing with large datasets, as it leverages the existing knowledge of the model and adapts it to changing data. Incremental re-training is particularly useful in online learning scenarios where the model continuously learns from incoming data streams. It helps maintain the model's relevance over time without the need for starting the training process from the beginning.

For Example: Spam mail classification with continuous learning.

Please refer to the reference links in the [References] section for more information.

Model Training and Run Best Practices

1. Trigger As Soon As Data is Available
    a. Once the data pipeline run is completed and data is validated properly, start the model run to keep up with the new data.
2. Keep Feature Engineering Automated
    a. While creating the model, automate feature engineering to streamline the process, enhance reproducibility, and adapt to changes in the dataset.

3. Consider a Shadow deployment as option
    a. If the model on Dev is updated with some new logics then enable shadow deployment where keep both model up and running on production but send 20% traffic of the model access to new model deployment.
    b. Test the model and if properly tested increase the traffic percent slowly here.
    c. Discard the old version at some point and make new model 100% up and running.
4. Set separate clusters just for model run if required compute is higher, and turn off the clusters after the run is completed. And continue on normal compute cluster other processed.
5. Set loggings of hyper parameters and model status.
    a. All major and effective hyper parameters and measurement metrics should be logged time to time.
    b. Monitor the model run based on current hyper parameter values and measurement metrics actively.
    c. Set triggers for the unwanted hyperparameter + metric values.
        i. For a complex model, there many parameters which shows if the model in stuck somewhere or not. In a complex model, there are various internal settings (hyperparameters) that affect how the model learns. These settings can give us clues about whether the model is getting stuck at some point during training.
        ii. Based loss curve of the model, and learning rate, whether the model is optimizing its output properly or not based on training data. By looking at the loss curve (a measure of how well the model is doing) and the learning rate (how fast the model is adjusting), we can figure out if the model is improving its predictions as it learns from the training data. This helps us ensure the model is working well and making progress.
    d. For forecasting models, if the model is forecasting at different grains, store the intermediate results.
        i. Skip the forecasting run if input Data Frame is invalid.
        ii. Keep model code to skip previous completed grain runs and start where it failed.
        iii. Create a detailed report of a grain forecast along with forecast values, differences in subsequent day/week/month forecast.
        iv. If ARIMA or other statistical models are used, log all the selected parameters values, trend, seasonality by the models. So if forecast looks wrong, over values or under values, then model output can be tweaked according to that.

Model Back Up Scheduling

Model backup scheduling is a fundamental practice within the realm of machine learning (ML) that involves systematically and regularly saving copies of ML models, including both their architectures and weights. This process is essential for preserving the integrity and evolution of machine learning models over time. The importance of model backup scheduling becomes particularly evident in scenarios where models are continuously refined, updated, or deployed for real-world applications. By adhering to a well-defined schedule, organizations ensure the availability of past model versions, offering the flexibility to roll back to prior iterations if needed. This not only provides a safety net against unexpected issues or errors introduced during model updates but also facilitates reproducibility in research and compliance with regulatory requirements. Moreover, model backup scheduling is critical for maintaining a historical record of model performances, insights gained, and improvements made, contributing to the transparency and accountability of machine learning initiatives. In the broader context of ML input and output data, model backup scheduling acts as a strategic safeguard, enhancing the resilience and reliability of machine learning systems by preserving the knowledge embedded in their evolving models.

Please refer to the section [Appendix A [Machine Learning Model Backup Methods]] for model backup methods. Automation of the model backup can be integrated in the code it-self. The backup code can be scheduled on timely basis (daily/weekly/monthly).

Model Back Up Scheduling Best Practices

Please refer to the appendix [Appendix A [Machine Learning Model Backup Methods] for knowing methods of taking model backups.

1. Models like Logistic Regression and other classification based statistical models, can be backed up once, with model version.
    a. These models can be exported or saved as .pickle format.
    b. With every run, the model file should be backed up with latest version. Version can be a datetime stamp of the backup time.
    c. There should be clean up/retention policy decided to keep N numbers of models only in the backup.
        i. Last run model should be stored as a Hot tier.
        ii. All other backup files should be stored in cool tier.
        iii. All very old backup should be stored/updated as Archive tiers.
2. Deep Learning models should be backed up along with their log files. All deep learning models can be stored in format like md5 where training weights, with each iteration parameter updates and loss is also stored.
    a. All DL models should be backed up as soon as the training is over. Naming of the files can be suffixed with timestamp along with the iterations.
    b. Based on the loss of the training, if the model takes another direction of training than the required one, then older model should be restored on quick basis.
        i. Last back up of every model should be in Hot tier.

ii. All the backups should be in cool tier.
iii. Very old backups should be in archive tiers.

Data Back Up Scheduling

Data backup scheduling is a crucial component of machine learning (ML) workflows, encompassing the systematic and regular process of creating copies of ML input and output data at predefined intervals. This practice ensures the preservation of valuable data sets, training data, model outputs, and associated metadata. Data backup scheduling is integral to mitigating the risk of data loss, which can have profound consequences on the accuracy and effectiveness of machine learning models. By adhering to a well-defined backup schedule, organizations safeguard against unforeseen events, such as hardware failures, system crashes, or accidental deletions, which could otherwise jeopardize critical data assets. In the context of machine learning, where the quality and availability of data profoundly influence model performance, a robust data backup strategy is imperative. It not only facilitates the recovery of datasets for retraining models but also contributes to the overall resilience of ML pipelines, ensuring continuity in data-driven decision-making processes. Thus, data backup scheduling serves as a proactive and strategic measure, preserving the integrity and accessibility of ML input and output data to uphold the reliability and effectiveness of machine learning initiatives. Ideally, Data Back Up is required where the output is generated as a file (i.e., Regression models, forecasting models etc.)

Data Back Up Scheduling Best Practices

1. Input Data back up should be taken timely basis. If all input data pull or ingestion is scheduled and a back up of older input data can be configured along with that only.
2. If Input data is incremental, (i.e., In Azure Cognitive Search, some data source are having files which can be updated or modified, added anytime by the users), then in that case, we need to take backup on timely basis at a particular frequency.
    a. A backup can be scheduled on daily basis if data-source have frequent updates.
    b. Otherwise, back up can be scheduled on weekly basis.
3. Output files are generated mostly when scheduled model is run. So, output files backup must be scheduled along with model runs.
4. Interactive model which are deployed and continuously interactive with users, for those, outputs are generated and stored in very frequent manner. For these models, the outputs and user feedback can be again used to re-train the models and for other analysis purposes.
    a. Back up of such model outputs should be scheduled via default mechanisms only.
    b. If database is used to store the interactive information, then enable the database backup mechanism on a particular frequency.
    c. Have Geo replication enabled for such models, and outputs.
5. If more output data is getting generated, then backup should be taken as parquet or some other compressed format to save the storage cost.
6. All these backups are very less accessed, these backups must be enabled for cool tier storage, instead of hot tier.
7. Very old backups, must be either Archive tier if those may be required in future.
8. For the models, where older backups are not required, such forecast models generated forecast of last year.

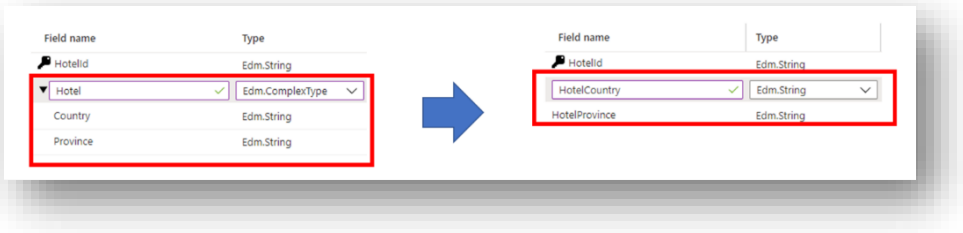
a. These forecast can be not necessary for the analysts, and can be deleted time to time. Scheduled data clean up activity should be there to clean up the data.

Concerned Model Best Practices

Azure AI Models

Indexes

1. Keep the index as small as possible.
    a. Large index size leads to slow query performance.
    b. Restructure the data to have smaller index fields as possible.
2. Choose the index attributes very carefully.
    a. Choose the attributes values like filterable, retrievable, searchable, facetable and - other attributes carefully. As these also impact the query performance directly.
    b. Choose searchable and retrievable very carefully as, answer curation depends on these fields.
3. Use simpler data types as much as possible.
    a. Choose simple and basic data types like string, integer etc., over complex datatype.



4. Keep monitoring index size at particular time interval, upgrade the capacity of the resource if required.
    a. Standard S1 cognitive search resource provides 25 GB 12 partitions. Total 300 GB, with 50 index creation limit.
    b. If data ingested or sourced gets higher than this, then upgrade the resource to Standard S2 or higher version. Check for Cost of this.

Indexer

1. Select indexer run time properly.
    a. Choose the timing considering priority, data availability, cost.
    b. Storage access read requires costing. If run on daily basis where data pipeline refresh is weekly , will incur irrelevant cost.
2. Monitor the scheduled run on timely basis.
    a. Indexer run will fail if ingested data does not match the index schema , datatype is updated or other reasons.
    b. If indexer is connected to a datasource and an index, if that source has other files which does not follow that index schema then it will fail.

    c. Setup alert mechanism based on indexer run.
3. Choose skillset if data is having complex format.
    a. Create a skillset which is defined for the complex data.
    b. Choose the skill set created for the datasource.
    c. If data follows default format provided by Azure (given below) then choose those.
        i. JSON line
        ii. JSON array
        iii. JSON object
        iv. CSV/Delimited Text
        v. Text
        vi. Default
    d. Check every run to see if count of documents are properly updated.
    e. Schedule multiple indexer if multiple datasources are associated.

Appendix A [Machine Learning Model Backup Methods]

Save the Model with .pickle

1. Pickle is a serialized format for python object.
2. It allows to serialize Machine Learning algorithm and save it in a file format.
3. It ideally follows .pickle extension but it can save model in other extensions like .sav (Statistical Package for the Social Science).
4. This further can be deserialized when required and can be used back to generate the output.
5. Dump() function is used to save the file, and load() function is used to restore the file.

Example:

Dump() Function

```
# Save Model Using Pickle
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
import pickle
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)
# Fit the model on training set
model = LogisticRegression()
model.fit(X_train, Y_train)
# save the model to disk
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

Load() Function

```
# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, Y_test)
print(result)
```

Saving Models using Joblib

1. Joblib is library provided in Scipy.
2. Models which use numpy objects, can be easily stored with this library.
3. Models which require lot of parameters or storing of entire dataset (K- Nearest Neighbour) can be stored be this.

Example

Dump() method and Load method() of Joblib

```
from joblib import Parallel, delayed
import joblib


# Save the model as a pickle in a file
joblib.dump(knn, 'filename.pkl')

# Load the model from the file
knn_from_joblib = joblib.load('filename.pkl')

# Use the loaded model to make predictions
knn_from_joblib.predict(X_test)
```

Storing Deep Learning Model

1. Keras separates the functionality of saving the machine learning model architecture and machine learning model weights.
2. The files stored all the weights of the last training iterations for all the layers of the model.
    a. The file is majorly stored in .h5 format.
    b. The format is HDF5 which stands for Hierarchical Data Format Version 5.
    c. It is designed to store and organize large amounts of heterogeneous data in a portable and scalable manner. HDF5 is commonly used in scientific and engineering applications where complex data structures, numerical datasets, and metadata need to be stored and shared efficiently.
    d. This file can be loaded back to incrementally train the model or even to generate output again.
    e. If newly trained model is diverged from the required path, then older stored .h5 file can be restored and new training can be started again for performance improvement.
3. Model architecture can be stored in JSON or YAML format.

Example

Model Saving to JSON

```
# serialize model to JSON
model_json = model.to_json()
```

```
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

Model Loading from JSON

```
# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
```

Model Saving to YAML

```
# serialize model to YAML
model_yaml = model.to_yaml()
with open("model.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

Model Loading from YAML

```
# load YAML and create model
yaml_file = open('model.yaml', 'r')
loaded_model_yaml = yaml_file.read()
yaml_file.close()
loaded_model = model_from_yaml(loaded_model_yaml)
# load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
```

References

Machine Learning Model Training: Link

Deep Learning Model Training with Databricks: Link

Training Machine Learning Models in Azure: Link

Train Keras Model in Azure: Link

Retraining the Models in Azure: Link

Best Practices: Link, Link