# PRJ-MLS-044: Automated Hyperparameter Tuning

**Certification:** AWS Certified Machine Learning – Specialty
**Domain:** Model Tuning and Optimization

---

## 1. Project Overview

This project demonstrates how to use **Amazon SageMaker's Automatic Model Tuning** to find the best version of a machine learning model. The performance of an ML model is highly dependent on the values of its **hyperparameters**—settings like learning rate, batch size, or the number of layers in a neural network. Finding the optimal combination of these hyperparameters can be a tedious and time-consuming process of trial and error.

SageMaker's hyperparameter tuning automates this process. It intelligently launches multiple training jobs in parallel, each with a different combination of hyperparameter values. It then uses sophisticated search strategies (like Bayesian optimization) to learn from the results of previous jobs and decide which combination to try next. This approach is far more efficient than random or grid searches and allows data scientists to quickly converge on the best possible model for their data.

### Key Objectives

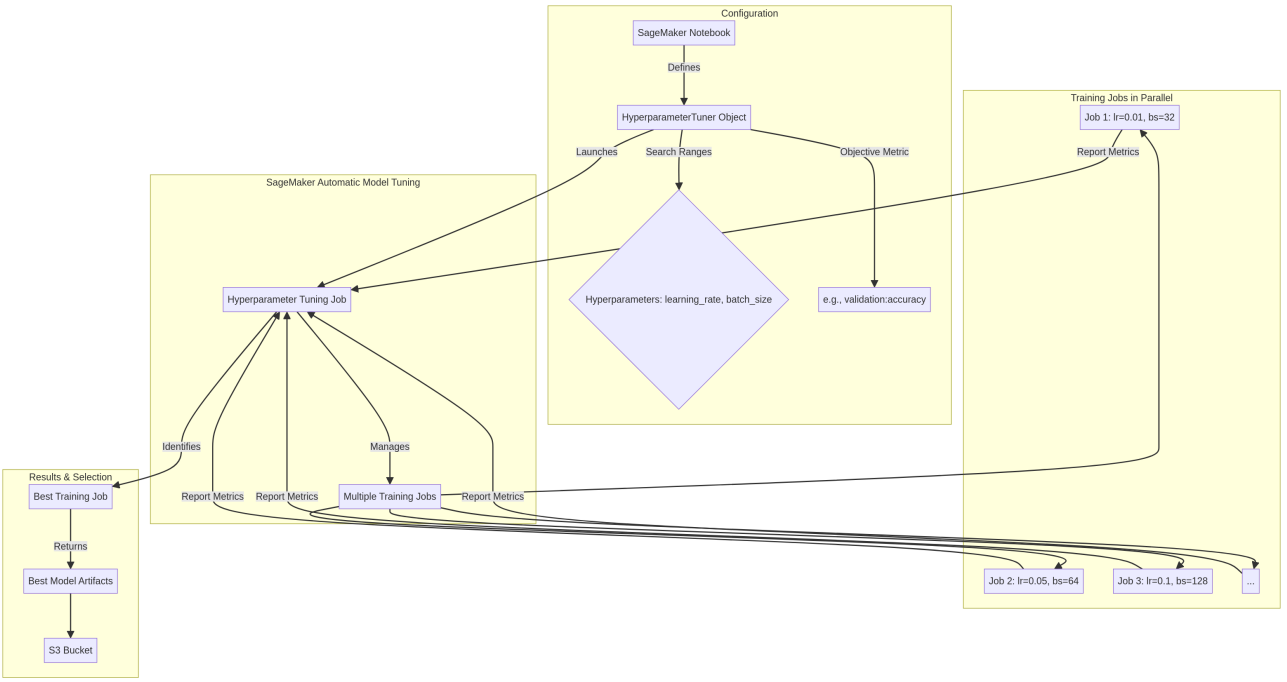
- Understand the importance of hyperparameter tuning in the ML lifecycle.
- Define a hyperparameter search space with various ranges (continuous, integer, categorical).
- Configure a SageMaker `HyperparameterTuner` object, specifying the objective metric to optimize.
- Launch a hyperparameter tuning job that runs multiple training jobs in parallel.
- Monitor the tuning job's progress and analyze the results.

- Identify and deploy the best model found by the tuning job.

---

## 2. Architecture

The architecture is managed by SageMaker, which orchestrates the parallel execution of many individual training jobs.




### Tuning Workflow:

1. **Configuration:**
    - In a **SageMaker Notebook**, the user defines a `HyperparameterTuner` object.
    - This object is configured with:
        - A standard SageMaker `Estimator` (e.g., for XGBoost, TensorFlow, etc.).
        - The **hyperparameter search ranges** (e.g., `learning_rate` from 0.01 to 0.1).
        - The **objective metric** to optimize (e.g., `validation:accuracy`). This metric must be printed to the logs by the training script in a specific format that SageMaker can parse.

        - The tuning strategy (e.g., Bayesian, Random).
        - Resource limits, such as the maximum number of parallel jobs and the total number of jobs to run.

2. **Tuning Job Execution:**
    - When the tuning job is launched, SageMaker's backend service takes over.
    - It starts launching individual **training jobs** in parallel, up to the configured limit.
    - For each job, it chooses a unique combination of hyperparameters from the defined search space.

3. **Metric Reporting and Optimization:**
    - As each training job runs, it computes and logs the objective metric (e.g., after each epoch).
    - SageMaker's tuning service scrapes these logs, collecting the performance of each hyperparameter combination.
    - Using this information, the Bayesian optimization algorithm builds a surrogate model to predict which hyperparameter combinations are likely to perform well. It then launches new training jobs to explore these promising areas of the search space.

4. **Results and Selection:**
    - After all training jobs are complete, the tuning job is marked as `Completed`.
    - SageMaker presents a ranked list of all the training jobs, ordered by their final objective metric score.
    - The **Best Training Job** and its corresponding model artifacts (the best model) are clearly identified and can be easily deployed.

---

## 3. Prerequisites

- An AWS account with administrative permissions.
- A SageMaker Notebook instance.

- A training script and a dataset in S3.
- The training script must be modified to print the objective metric that SageMaker will track.

---

## 4. Step-by-Step Implementation Guide

### Step 4.1: Prepare the Training Script

Your training script must print the objective metric in a format that SageMaker can parse. This is typically done by printing a string that matches a regular expression.

**Example (in your evaluation loop):**

```
# ... after calculating validation accuracy ...
print(f"[Validation] Accuracy: {accuracy}")
```

### Step 4.2: Configure the HyperparameterTuner

In your SageMaker Notebook:

1. **Create a standard Estimator:**

```
from sagemaker.estimator import Estimator


xgb_estimator = Estimator(
    image_uri=sagemaker.image_uris.retrieve("xgboost", region, "1.5-
1"),
    role=role,
    instance_count=1,
    instance_type="ml.m5.large",
    output_path=f"s3://{bucket}/output"
)
```

2. **Define Hyperparameter Ranges:**

```
from sagemaker.tuner import IntegerParameter, ContinuousParameter,
CategoricalParameter


hyperparameter_ranges = {
    "eta": ContinuousParameter(0.1, 0.5), # learning rate
    "max_depth": IntegerParameter(2, 6),
    "num_round": IntegerParameter(50, 200),
    "subsample": ContinuousParameter(0.5, 1.0)
}
```

3. **Define the Objective Metric and Regex:**

```
objective_metric_name = "validation:accuracy"
objective_type = "Maximize"
metric_definitions = [{
    "Name": objective_metric_name,
    "Regex": "\[Validation\] Accuracy: (.*?)"
}]
```

4. **Create the `HyperparameterTuner` Object:**

```
from sagemaker.tuner import HyperparameterTuner


tuner = HyperparameterTuner(
    estimator=xgb_estimator,
    objective_metric_name=objective_metric_name,
    hyperparameter_ranges=hyperparameter_ranges,
    metric_definitions=metric_definitions,
    max_jobs=20, # Total number of jobs to run
    max_parallel_jobs=4, # Number of jobs to run in parallel
    objective_type=objective_type,
    strategy="Bayesian"
)
```

### Step 4.3: Launch the Tuning Job

```
tuner.fit({"train": "s3://my-bucket/my-data/train/", "validation": "s3://my-bucket/my-data/validation/"})
```

- This command will start the tuning job. You can monitor its progress in the SageMaker console under **Hyperparameter tuning jobs**.
- The console provides a detailed view of all the training jobs launched, their hyperparameter values, and their corresponding objective metric scores.

### Step 4.4: Analyze Results and Deploy the Best Model

1. **Wait for Completion:** The tuning job will take some time to complete all 20 training jobs.
2. **Analyze Results in a Notebook:**

   ```
   # Attach to the completed tuning job
   tuner_results =
   HyperparameterTuner.attach(tuner.latest_tuning_job.name)

   # Get a dataframe with the results
   results_df = tuner_results.analytics().dataframe()
   print(results_df.sort_values("FinalObjectiveValue", ascending=False))
   ```

   This will give you a clear, ranked list of the performance of all hyperparameter combinations.
3. **Deploy the Best Model:**
   - The `HyperparameterTuner` object makes it easy to deploy the best model found.

   ```
   best_model_predictor = tuner.deploy(
       initial_instance_count=1,
       instance_type="ml.m5.large"
   )
   ```

This command finds the best training job, takes its model artifacts, and deploys them to a SageMaker real-time endpoint.

---

## 5. Advanced Features

- **Warm Start:** You can “warm start” a new tuning job using the results of a previous one. This allows you to continue searching the hyperparameter space without starting from scratch.
- **Early Stopping:** SageMaker can automatically stop training jobs that are not performing well, saving time and money. If a new job isn’t improving on the metric after a few epochs, SageMaker will terminate it and start a new one with different hyperparameters.

---

## 6. Cleanup

1. **Delete the Endpoint:**

```
best_model_predictor.delete_endpoint()
```

2. **Clean up S3 Artifacts:** Delete the model artifacts and outputs from the S3 bucket.
3. The individual training jobs and the hyperparameter tuning job itself are metadata and do not incur costs, but you can delete them from the console for cleanliness.

## Business Context

### The Problem

Organizations want to leverage AI/ML for business insights but lack the expertise to build secure, scalable ML systems. ML projects fail due to poor data quality, model

drift, and lack of monitoring. Security and compliance requirements for ML systems are often overlooked.

### The Solution

Production-ready ML system with automated data pipelines, model training, deployment, and monitoring. Implements MLOps best practices with version control, A/B testing, and automated retraining. Ensures data privacy and model security throughout the ML lifecycle.

### Business Value

- **Business Intelligence:** AI-driven insights improve decision-making and outcomes
- **Operational Efficiency:** Automated ML workflows reduce manual data science effort by 60%
- **Model Reliability:** Continuous monitoring detects and corrects model drift automatically
- **Compliance Assurance:** Built-in controls for data privacy and model governance

### Risk Mitigation

Protects sensitive data in ML pipelines, prevents biased or inaccurate models, ensures model explainability, and maintains compliance with AI regulations.

## GRC Mapping

### Compliance Frameworks

- **NIST AI RMF:** Trustworthy and responsible AI
- **ISO/IEC 23894:** AI risk management
- **NIST CSF:** PR.DS-2 (Data in transit), PR.DS-5 (Data leak protection)
- **Responsible AI Principles:** Fairness, accountability, transparency, ethics

## Security Controls Implemented

- Data anonymization and pseudonymization
- Model explainability and interpretability
- Bias detection and mitigation
- Model versioning and rollback
- Automated model monitoring and alerting

## Audit Evidence

- Model cards documenting intended use and limitations
- Bias and fairness evaluation reports
- Model performance metrics over time
- Data processing and transformation logs

## Regulatory Alignment

- **GDPR:** Article 22 (Right to explanation), Article 25 (Privacy by design)
- **AI Act (EU):** Transparency and accountability requirements
- **CCPA:** Data privacy for ML training data
- **SOC 2:** CC6.1 (Access to sensitive data)