Extreme Precipitation Prediction MLOps System
AAI-540 | Group 4
Ahmed Salem & Victor Salcedo
AWS SageMaker Implementation Demo
Hi. This is our Extreme Precipitation Prediction MLOps system.
Our objective was to move beyond model training and implement a production-ready machine learning system in AWS SageMaker capable of detecting extreme rainfall events using NOAA weather station data.
This project validates infrastructure, monitoring, CI/CD governance, and deployment readiness.

Production System Validation & MLOps Workflow
Feature Store & Feature Groups
Infrastructure Monitoring
Model & Data Monitoring Reports
CI/CD Pipeline (Success & Failure)
Model Registry
Batch Inference Outputs
In this demonstration, we will validate the full ML lifecycle.
We will walk through:
Feature Store and feature groups
Model artifact packaging and deployment
Monitoring baseline generation
Infrastructure monitoring via CloudWatch
CI/CD pipeline execution in both successful and failed states
Model registry integration
And inference readiness
This is a system-level validation, not just a model training demo.

This diagram represents our end-to-end ML system.
We begin with NOAA GHCN daily weather data stored in Amazon S3.
We preprocess and engineer features such as lagged precipitation, rolling aggregates, and seasonal indicators.
These features are stored in SageMaker Feature Store to ensure reproducibility and schema consistency.
Models are trained using SageMaker, and artifacts are stored in S3.
Approved models are deployed to SageMaker endpoints.
Model Monitor generates baselines and schedules recurring monitoring jobs.
CloudWatch captures metrics and triggers alarms if data constraints are violated.
This architecture ensures controlled training, deployment, monitoring, and retraining capability.

Feature Store – Create Feature Group
We create a SageMaker Feature Group with both online and offline stores enabled.

This ensures time-aware versioning and reproducibility across training and inference workflows.
Feature management is foundational to production ML.

Feature Store – Verify Configuration
Here we validate the Feature Group ARN, schema, and S3 storage location.
This confirms that our feature layer is operational and correctly configured.

Model Artifact Upload to S3
After training, the model and inference script are packaged into a model.tar.gz artifact.
This artifact is uploaded to S3 and becomes the deployable unit for SageMaker.
This separates training logic from deployment logic.

Deploy Model Endpoint
We deploy the model to a SageMaker endpoint using an ml.m5.large instance.
Successful deployment confirms inference readiness.
This endpoint can be integrated into downstream alerting or forecasting systems.

Generate Monitoring Baseline
We generate baseline statistics from the training dataset.
These baselines define expected feature distributions and constraint thresholds.
They serve as the foundation for drift detection.

Configure Model Monitor
We configure SageMaker Model Monitor with defined runtime and storage parameters.
This automates data quality validation against the established baseline.
Monitoring is scheduled, not manual.

Schedule Monitoring Job
We create a recurring monitoring schedule to continuously evaluate inference data.
This operationalizes drift detection within the system.

Create CloudWatch Alarm

We configure a CloudWatch alarm on the ConstraintViolations metric.
If thresholds are exceeded, alerts are triggered.
This integrates model monitoring with infrastructure observability.

Verify CloudWatch Alarm
We now move to CI/CD.
Here we confirm metric binding, threshold configuration, and namespace alignment.
This validates that infrastructure monitoring is active.

Pipeline Parameter Definition
We define a dynamic accuracy threshold within our SageMaker Pipeline.
This parameter governs automated model promotion decisions.
It enables flexible governance without rewriting pipeline code.

Define & Register Pipeline
We register a pipeline containing training, evaluation, and conditional approval steps.
This formalizes the ML workflow into a reusable CI/CD DAG.

Pipeline Execution – Success & Failure
This slide demonstrates a failed run demonstration.
In the successful run, the model meets performance thresholds and proceeds.
In the failure case, we tighten the threshold, and the model is blocked from promotion.
This validates automated quality gating and governance.

Risks & Future Improvements
Data drift across climate regions
Class imbalance challenges
Enhanced automated retraining
Advanced statistical drift detection
Before closing, it’s important to address operational risks and future scalability considerations.
From a data perspective, extreme precipitation events are inherently rare, which creates class imbalance risk. If climate patterns shift or station reporting quality degrades, the model could experience silent recall degradation. Geographic coverage bias is also a concern, since some regions have denser or more reliable station data than others.
From an infrastructure standpoint, the current system is optimized for batch inference at moderate scale. If we were to expand to near real-time regional alerting or significantly increase station coverage, we would need to evaluate endpoint auto-scaling, distributed training,

and more robust storage partitioning strategies.
From a governance perspective, drift detection is currently threshold-based. A production-grade system would incorporate statistical drift tests, automated retraining triggers, and controlled rollback mechanisms through a formal model registry approval workflow.
Future enhancements would include expanding feature windows to capture longer-term precipitation accumulation, evaluating gradient boosting or ensemble methods for improved recall, and implementing automated retraining pipelines triggered by CloudWatch drift events.
These improvements would transition this system from a validated academic MLOps implementation to a scalable, production-grade environmental risk intelligence platform.
Next up Ahmed will discuss model performance.

Model Performance — Baseline Evaluation

Production-Ready Evaluation with Time-Based Split

Logistic Regression Pipeline with Feature Engineering
Selected Threshold: 0.65 (Optimized for F1)
ROC-AUC: 0.72 | PR-AUC: 0.10
F1: 0.17 | Precision: 0.11 | Recall: 0.38
Recall prioritized due to rare precipitation events

```
y_probs = model.predict_proba(X_test)[:,1]
best_threshold = optimize_threshold(y_probs, y_test)
metrics = evaluate_model(y_test, y_probs, best_threshold)
```

From Infrastructure to Model Governance

Training and deployment infrastructure established
CI/CD pipeline successfully executed
Monitoring framework configured
Next: Performance validation and production controls

Up to this point, we have demonstrated the infrastructure components of the system, including feature storage, model deployment, monitoring configuration, and CI/CD pipeline execution. n the next section, we shift focus to model performance validation, threshold optimization, automated governance, and drift monitoring. These components transform the system from a technical implementation into a controlled and production-aligned MLOps workflow

Threshold Optimization & Validation Gate
Automated Quality Control in CI/CD Pipeline
Threshold search from 0.05 → 0.95

Best threshold selected by F1 maximization
Pipeline fails if F1 < 0.20
Pipeline fails if Recall < 0.50
Prevents weak models from reaching production

```
if f1 < min_f1 or recall < min_recall:
    raise ValueError("Model validation failed")
```

To improve classification performance under class imbalance, we conducted a systematic threshold search ranging from 0.05 to 0.95 rather than relying on the default 0.5 probability cutoff. The optimal threshold was selected based on F1 score maximization, allowing us to better balance precision and recall. Beyond threshold tuning, we implemented an automated validation gate within the CI/CD pipeline. If the model's F1 score falls below 0.20 or recall drops below 0.50, the pipeline execution fails. This governance mechanism ensures that only models meeting predefined quality standards are eligible for registration and deployment

Confusion Matrix – Test Set Analysis
Understanding Precision vs Recall Tradeoff
True Negatives: 5868
False Positives: 1078
False Negatives: 209
True Positives: 131
Balanced threshold tuning impacts FP/FN tradeoff
The confusion matrix illustrates the classification outcomes under the optimized threshold. The model correctly identified 131 true precipitation events while maintaining a high number of true negatives, which is expected given the imbalanced nature of the dataset. Although false positives are present, they represent a tradeoff inherent in rare-event detection. By tuning the threshold intentionally, we explicitly control the balance between missed events and false alerts. This approach embeds business-relevant decision-making directly into model evaluation

Model Monitoring – PSI Drift Detection
Automated Feature Stability Monitoring
PSI Threshold: 0.20
prcp_lag_1: 0.0003
prcp_roll_7: 0.0051
TMAX: 0.0081 | TMIN: 0.0055
Retrain Triggered: No (All values below threshold)

```
psi_value = calculate_psi(train_feature, production_feature)
if psi_value > 0.2:
    retrain_needed = True
```

Following deployment, we implemented Population Stability Index (PSI) to monitor feature distribution drift between training and production

data. A PSI threshold of 0.20 was defined to signal significant distributional change requiring retraining. All monitored features, including lagged precipitation and temperature variables, remained well below this threshold. As a result, no retraining was triggered during monitoring. This demonstrates a stable model environment and reflects production-grade model observability practices.

Batch Inference Workflow
Production Batch Scoring Simulation
Load registered model artifact
Generate probability-based predictions
Apply optimized threshold (0.65)
Export structured predictions for downstream systems

```
probs = model.predict_proba(X_batch)[:,1]
preds = (probs >= 0.65).astype(int)
df['prediction'] = preds
```

To simulate production batch scoring, we loaded the registered model artifact and generated probability-based predictions on new data. The optimized threshold of 0.65 was applied to convert probabilities into binary classifications. Structured outputs were exported for downstream system consumption, mirroring enterprise data workflows. This batch inference pipeline bridges model development and operational deployment. It ensures predictions are reproducible, traceable, and ready for integration into decision-support systems

End-to-End Reproducible MLOps Pipeline
Modular & CI/CD-Ready Architecture
Train → Evaluate → Validate → Deploy → Monitor
Independent script execution
Reproducible and production-aligned workflow

```
python train.py
python evaluate.py --min_f1 0.20 --min_recall 0.50
python inference.py
python monitor.py --psi_threshold 0.2
```

The pipeline was designed as a modular and script-driven architecture encompassing training, evaluation, validation, deployment, and monitoring. Each stage can be executed independently, supporting CI/CD integration and reproducibility. This separation of concerns reflects modern MLOps best practices. By structuring the system in this manner, we ensure maintainability, scalability, and operational consistency. The result is a production-aligned workflow rather than an isolated modeling experiment.

Future Enhancements & Production Scaling
Extending the MLOps System
Implement XGBoost or LightGBM for improved performance

Advanced statistical drift detection (KS-test, JS divergence)
Automated retraining when PSI exceeds threshold
Model performance dashboards & alerting
While the current system establishes a strong baseline, several enhancements could further improve performance and scalability. Advanced models such as gradient boosting algorithms may improve predictive accuracy under class imbalance. More sophisticated drift detection techniques, including statistical divergence measures, could strengthen monitoring. Automated retraining triggers and dashboard-based performance monitoring would enhance system responsiveness. These improvements would extend the framework into a fully automated, enterprise-scale MLOps solution.

References (APA 7)

Amazon Web Services. (2024). Amazon SageMaker Developer Guide.
National Oceanic and Atmospheric Administration (NOAA). (2024). GHCN-Daily Dataset.
OpenAI. (2024). ChatGPT (GPT-4). Used for formatting and code debugging support.