Mitigating Drift in Machine Learning Systems through Continuous Input Monitoring

Lucas Helfstein Rocha

Advisor: Kelly Rosa Braghetto

Master's Thesis Defense

Instituto de Matemática, Estatística e Ciência da Computação
Universidade de São Paulo

March / 2026

Presentation Outline

Outline

Context

- Machine learning systems are widely adopted across domains.
- Unlike traditional software, their behavior depends heavily on input data.
- After deployment, input data often changes over time.

Data changes may degrade model performance.

The Problem

- Distribution shifts may cause silent performance degradation.
- Traditional testing is insufficient after deployment.
- Continuous monitoring becomes essential.

ML systems must be monitored to remain reliable in dynamic environments.

Research Questions

RQ1: How should an ML system be structured to enable effective drift monitoring?

RQ2: How can drift monitoring improve model performance in production?

Objectives

This thesis investigates:

- How drift monitoring can be incorporated into ML systems.
- How monitoring affects system robustness.

Focus:

- Architectural organization
- Empirical evaluation of drift detection techniques

Outline

Machine Learning

Goal: learn a mapping from inputs to outputs.

$$g : X \to Y$$

Instance representation:

$$x = (a_1, a_2, \ldots, a_M)$$

- a_i = feature
- Features can be categorical or numerical

Supervised Classification

Training dataset:

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

Goal: learn a function that approximates the true relationship

$$g \approx f$$

Classification problem:

$$Y = \{Y_1, Y_2, \ldots, Y_K\}$$

Classifier prediction:

$$g(x) = \hat{Y}$$

Model Evaluation Setup

Dataset $\mathcal{D}$ of size N is divided into:

- $\mathcal{D}_{\text{training}}$ – used to train model g
- $\mathcal{D}_{\text{test}}$ – containing W labeled examples, used to evaluate performance

Accuracy measures correct predictions on the test set:

$$A(g) = \frac{1}{W} \left|\{(x, y) \in \mathcal{D}_{\text{test}} : g(x) = y\}\right| \tag{1}$$

Approximation, Generalization and Overfitting

Key concepts related to the behavior of classification models include:

Approximation

- Performance on training data

Generalization

- Performance on unseen data

Overfitting
Occurs when a model performs well on training data but poorly on unseen data.

Confusion Matrix

Performance metrics are based on:

- True Positives (TP)
- False Positives (FP)
- True Negatives (TN)
- False Negatives (FN)

	$g(x) = Y_k$	$g(x) \neq Y_k$
$y = Y_k$	TP	FN
$y \neq Y_k$	FP	TN

Evaluation Metrics

Precision

$$P = \frac{TP}{TP + FP} \tag{2}$$

Recall (Coverage)

$$C = \frac{TP}{TP + FN} \tag{3}$$

F1-score

$$F1 = \frac{2PC}{P + C} \tag{4}$$

Evaluation Metrics

ROC Curve (Receiver Operating Characteristic)
Represents the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) for different classification thresholds.

AUC (Area Under the ROC Curve)
Measures how well a classifier separates positive and negative classes.

- Derived from the ROC curve (TPR vs FPR)
- $AUC = 1 \rightarrow$ perfect classifier
- $AUC = 0.5 \rightarrow$ random classifier

Machine Learning System

A Machine Learning System is a software system that integrates an ML model into its architecture.

It involves more than training a model:

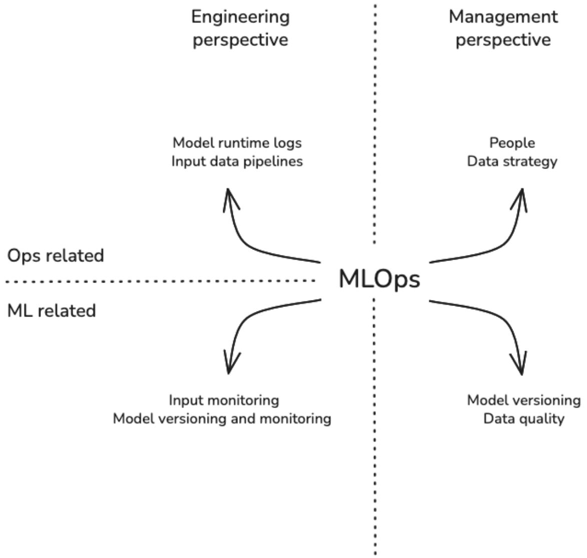
- Data pipelines
- Deployment
- Monitoring
- Governance

Lifecycle of a Machine Learning System

1. Data preparation
2. Model training and validation
3. Deployment in production
4. System data monitoring
5. Model performance monitoring

MLOps: Practices for building and operating machine learning systems and its lifecycle.

MLOps Conceptual Map



Outline

Data Drift

Training data for attribute a_i:

$$P^{a_i}(x), \quad F^{a_i}(x)$$

Incoming data in interval $[t, u]$:

$$P^{a_i}_{t,u}(x), \quad F^{a_i}_{t,u}(x)$$

Drift occurs when the distributions differ.

Numeric attributes:

$$F^{a_i}(x) \neq F^{a_i}_{t,u}(x)$$

Categorical attributes:

$$P^{a_i}(x) \neq P^{a_i}_{t,u}(x)$$

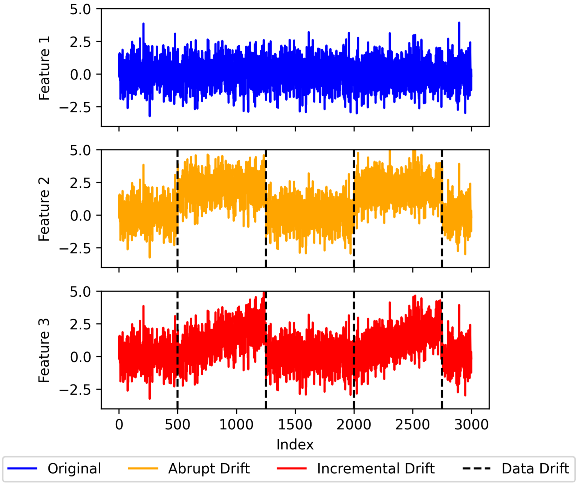
Intuition Behind Drift

Data drift occurs when the statistical properties of the input data change over time.

- Change in feature distribution
- Change in relative frequencies
- Change in cumulative behavior

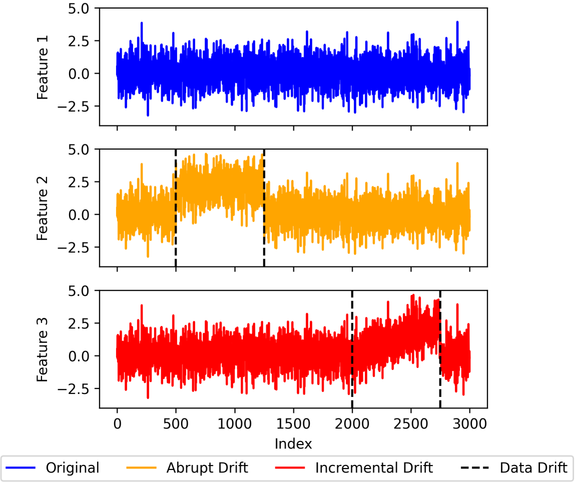
Impact: Potential degradation in model generalization.

Data Drift Patterns



Recurring drift: Distribution shifts and later returns to previous state.

Data Drift Patterns



Switching drift: Alternating distributions across time windows.

Concept Drift

Concept drift occurs when the relationship between inputs and the target variable changes over time.

Training distribution:

$$P(X, Y)$$

Production distribution:

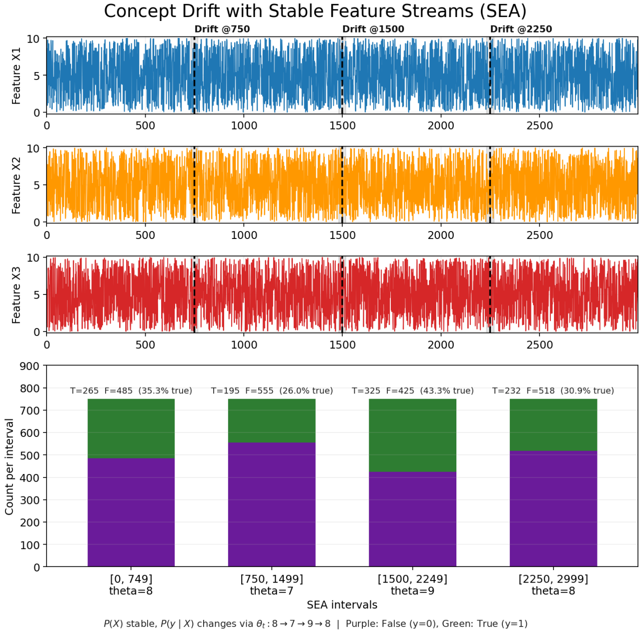
$$P_t(X, Y)$$

Formal Definition
Concept drift occurs if there exists a time *t* such that:

$$P_{0,t}(X, Y) \neq P_{t+1,u}(X, Y), \quad u > t$$

Meaning: the joint distribution changes over time.

Concept Drift



Error-Rate Based Drift Detection

Idea: Monitor model performance over time.

If the prediction error rate changes significantly $\Rightarrow$ trigger a drift alarm.

Common Algorithms

- Drift Detection Method (DDM)
- Early Drift Detection Method (EDDM)
- Adaptive Windowing (ADWIN)

These methods detect statistical changes in prediction errors.

Data Drift Detection

Goal: Monitor divergence between reference and current data distributions.

In this work, we consider two categories:

- Statistical methods
- Distance-based methods

Statistical Methods

Compare two datasets using hypothesis testing.

Output:

- Test statistic
- p-value

Low p-value $\Rightarrow$ distributions differ $\Rightarrow$ drift

Kolmogorov–Smirnov Test

Given empirical distributions F and G:

$$KS(F, G) = \sup_x |F(x) - G(x)|$$

Reject H_0 if:

$$KS(F, G) > c(\alpha)\sqrt{\frac{n+m}{nm}}$$

Multiple KS + Bonferroni Correction

Apply KS test independently across d dimensions.

Control Type I error using:

$$\alpha' = \frac{\alpha}{d}$$

Reject if:

$$\min_{k=1,\ldots,d} KS(F_k, G_k) > c\left(\alpha'\right)\sqrt{\frac{n+m}{nm}}$$

Distance-Based Methods

Measure dissimilarity directly between distributions.

Output:

- Numerical distance

Larger distance $\Rightarrow$ larger distributional shift

KL and JS Divergence

Kullback–Leibler Divergence

$$KL(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

Jensen–Shannon Divergence

$$JS(P||Q) = \frac{1}{2} KL \left(P|| \frac{P+Q}{2} \right) + \frac{1}{2} KL \left(Q|| \frac{P+Q}{2} \right)$$

Hellinger Distance

$$H(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{n} (\sqrt{p_i} - \sqrt{q_i})^2}$$

Symmetric and bounded measure of divergence.

Outline

Drift Detection Methods

Drift detection requires:

- A divergence measure
- A decision mechanism (threshold or statistical test)

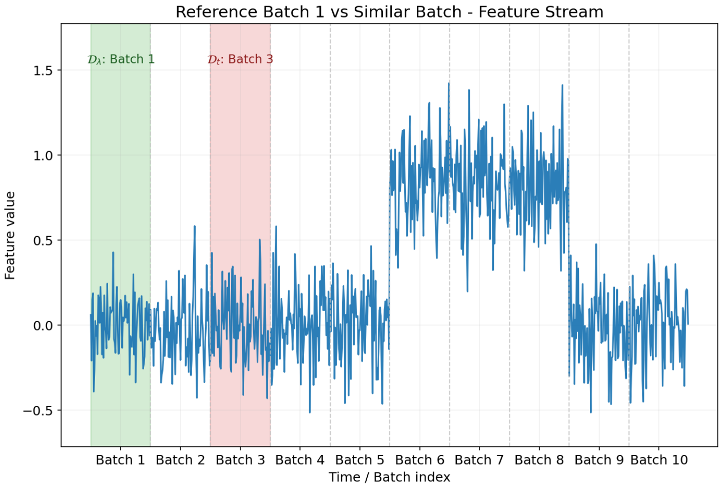
In this work we evaluate:

- HDDDM (Hellinger distance)
- JSDDM (Jensen–Shannon divergence)
- KSDDM (Kolmogorov–Smirnov test)

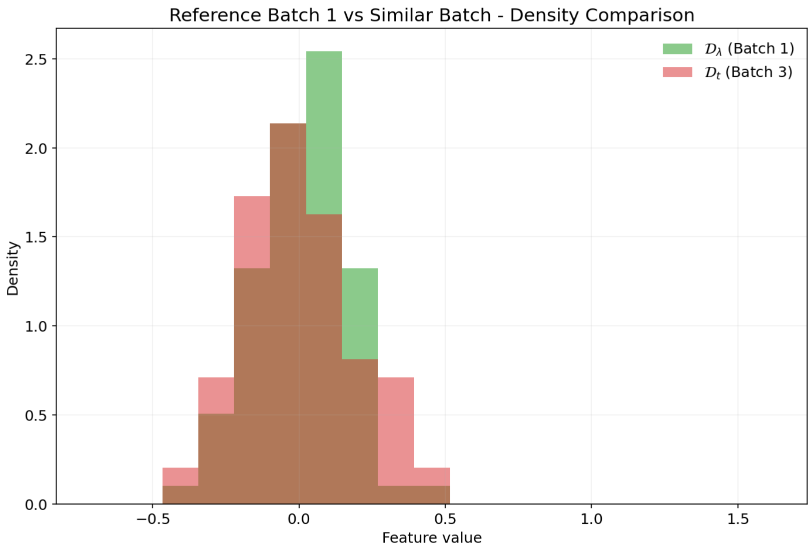
Common assumptions

- Classifier-free methods (operate on raw features)
- Incremental batch setting
- Compare current batch $\mathcal{D}_t$ with reference $\mathcal{D}_\lambda$

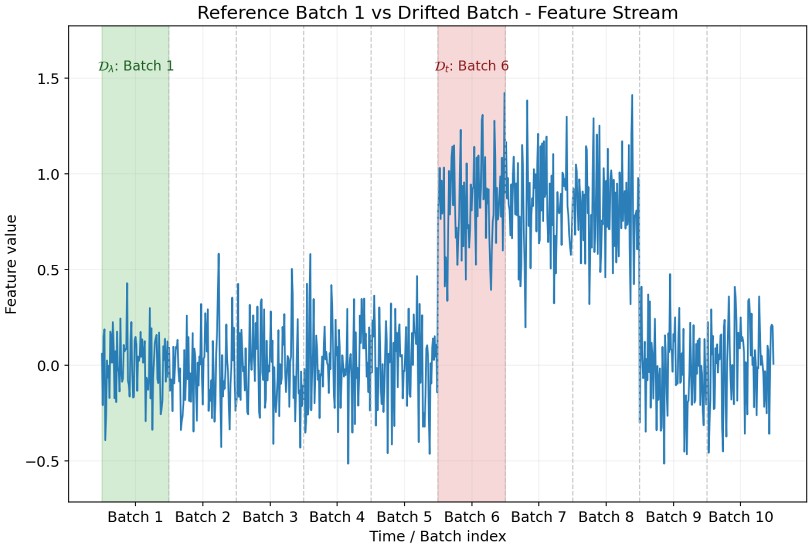
Drift Detection Methods



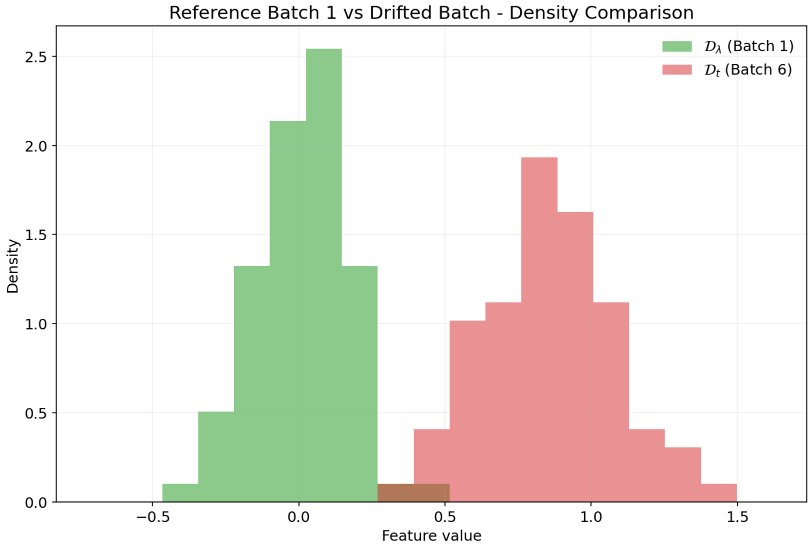
Drift Detection Methods



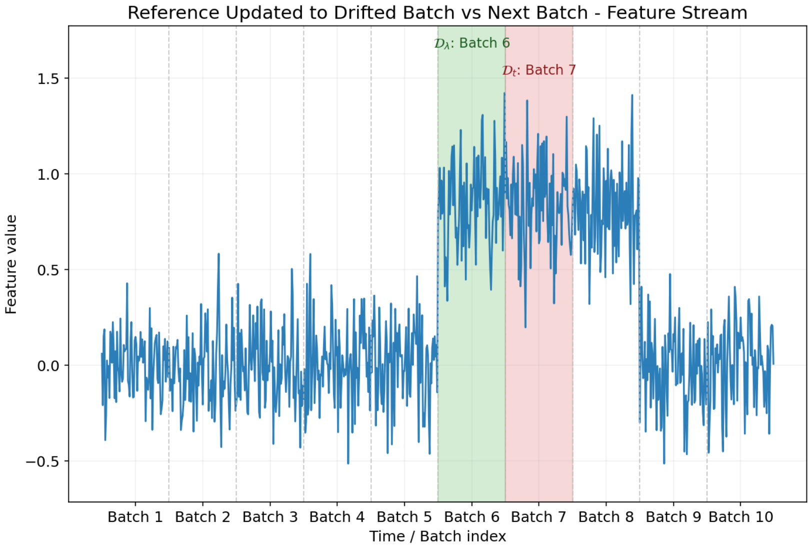
Drift Detection Methods



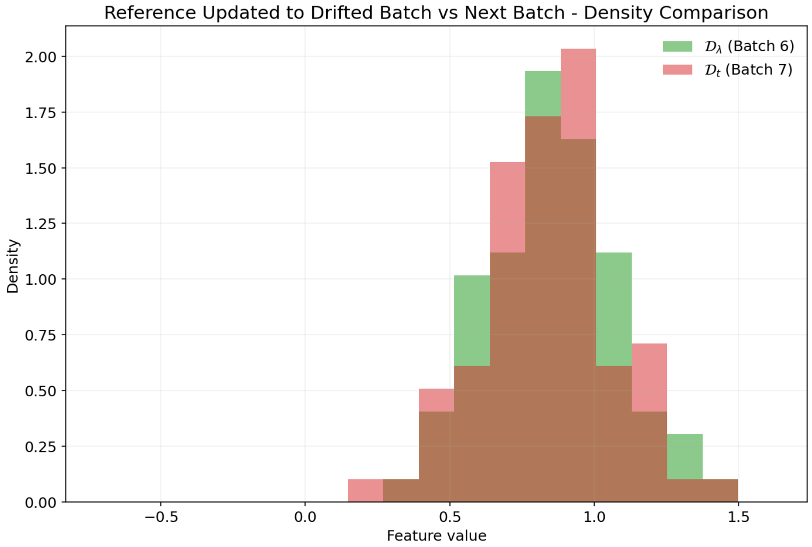
Drift Detection Methods



Drift Detection Methods



Drift Detection Methods



Hellinger Distance Drift Detection Method - HDDDM

At each time t:

- Build histograms from $\mathcal{D}_t$ and $\mathcal{D}_\lambda$
- Compute average Hellinger distance:

$$\delta_H(t)$$

- Compute change:

$$\epsilon(t) = \delta_H(t) - \delta_H(t-1)$$

Drift if:

$$|\epsilon(t)| > \beta(t)$$

HDDDM Decision Rule

Adaptive threshold:

$$\beta(t) = \hat{\epsilon} + \gamma\hat{\rho}$$

- $\hat{\epsilon}$: mean of past changes
- $\hat{\rho}$: standard deviation of changes
- γ: sensitivity parameter

Drift condition:

$$|\epsilon(t)| > \beta(t)$$

Drift is detected when the observed change exceeds the expected variation.

Jensen-Shannon Drift Detection Method - JSDDM

Extension of HDDDM:
Replace Hellinger distance with Jensen–Shannon divergence.

$$\delta_{JS}(t) = \frac{1}{d} \sum_{k=1}^{d} JS(P_k || Q_k)$$

Decision rule identical to HDDDM.

Kolmogorov-Smirnov Drift Detection Method - KSDDM

For each feature k:

- Compute KS statistic between $\mathcal{D}_t$ and $\mathcal{D}_\lambda$

$$\delta_{KS}(t) = \min_{k=1,\ldots,d} KS(F_k, G_k)$$

Drift if:

$$\delta_{KS}(t) > c\left(\alpha'\right)\sqrt{\frac{n+m}{nm}}$$

Outline

Research Methodology

Goal: identify concepts, tools, and practices related to Software Engineering for Machine Learning.

Literature Review Strategy

- Multivocal literature review
- Academic sources:
  - IEEE Xplore
  - ACM Digital Library
  - Google Scholar
- Grey literature:
  - Industry engineering blogs
  - Technical articles and reports
  - Documentation of MLOps tools

Sources were selected based on relevance to **ML lifecycle, monitoring, deployment, and experimentation**.

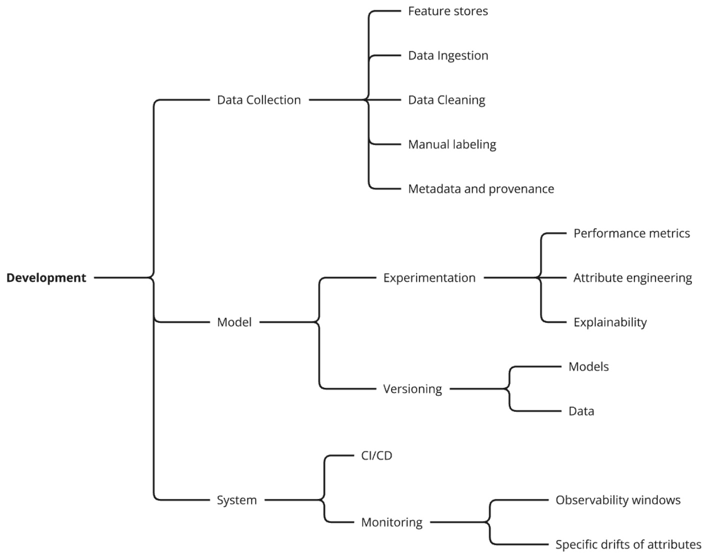
Taxonomy Structure

The taxonomy is organized according to:

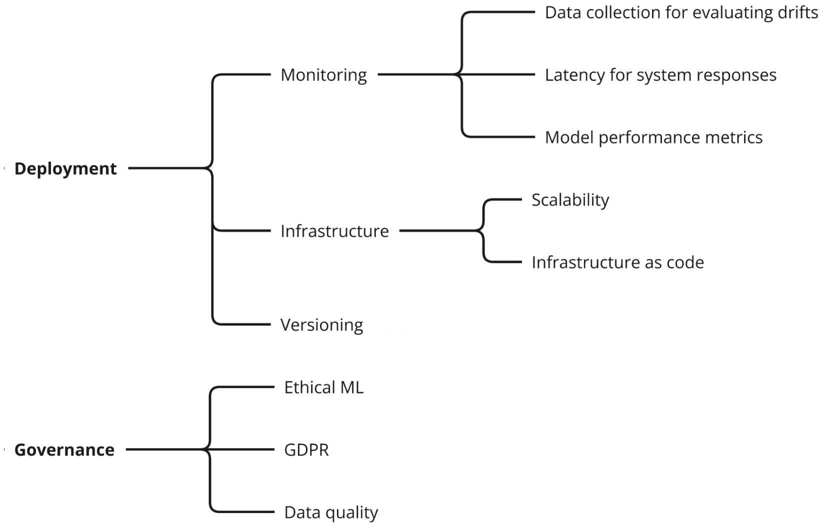
- Data Science concerns
- ML Engineering concerns
- Data Engineering concerns
- Data Governance concerns

Purpose: Provide a structured view of ML system components.

Taxonomy of ML System Concepts



Taxonomy of ML System Concepts



Software Engineering for ML Systems

Prior research highlights:

- Technical debt and hidden dependencies in ML systems (Sculley et al. 2015)
- Need for continuous testing and monitoring (Breck et al. 2017; Schröder and Schulz 2022)
- Architectural challenges in integrating ML components (Serban and Visser 2022; Kreuzberger et al. 2023)

Recent studies propose taxonomies and MLOps frameworks, but detailed architectural decompositions remain limited.

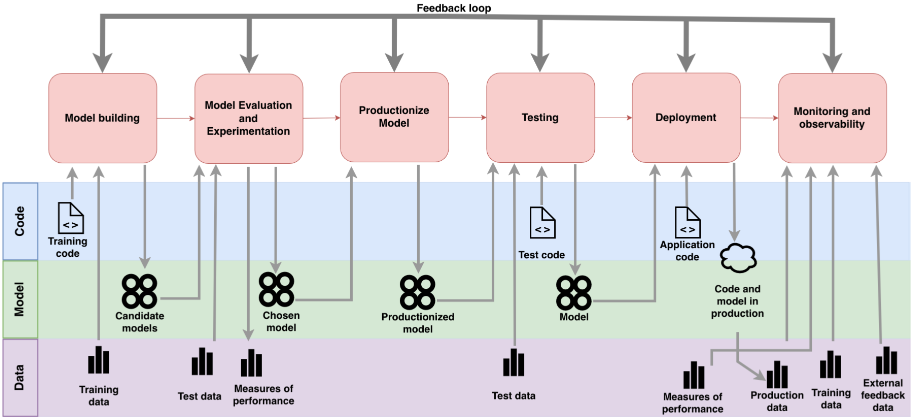
Drift Detection in ML Systems

Drift detection research focuses on:

- Concept drift in streaming environments (Lu et al. 2018; Gama and Castillo 2006)
- Distance-based input monitoring (e.g., Hellinger, KL) (Ditzler and Polikar 2011; Dasu et al. 2006)
- Statistical two-sample testing (Rabanser et al. 2019)
- Empirical comparisons of detection reliability (Rabanser et al. 2019)

Most approaches emphasize algorithms rather than system-level architectural integration (Sculley et al. 2015).

Feedback Loop Concept[1]



[1]Sato et al. 2019.

Feedback Loop Concept

- Information flows through:
  - Data
  - Model
  - Code
- Not strictly sequential

Monitoring may trigger:

- Retraining
- Parameter adjustments
- Infrastructure changes

Identified Gap

The literature provides:

- Architectural perspectives for ML systems
- Drift detection algorithms

However, there is limited work that:

- Integrates monitoring as a structured architectural component
- Connects drift detection techniques to MLOps workflows

This motivates defining architectural requirements for ML systems.

Requirements for ML System Architectures

To support reliable ML systems, an architecture should provide:

- **Lifecycle traceability** across training, deployment, and monitoring
- **Monitoring mechanisms** for input data and model performance
- **Feedback loops** connecting production signals to development
- **Reproducibility** of experiments and model versions
- **Integration with MLOps workflows**

Motivation: Enable continuous monitoring and systematic model evolution.

Outline

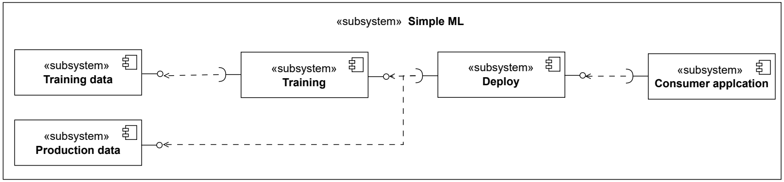
Motivation for Structuring ML Systems

In practice, many ML systems are built without a clear architectural structure.

- ML pipelines evolve across multiple lifecycle stages
- Monitoring is often added in an ad-hoc manner
- Lack of traceability increases technical debt
- Feedback loops are rarely formalized

Goal: Structure the lifecycle to enable systematic monitoring and feedback.

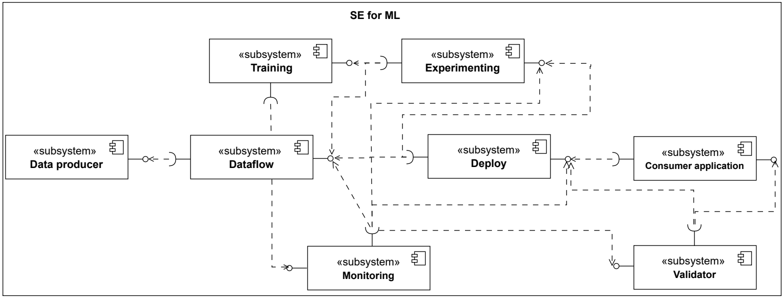
Simple ML Architecture


«subsystem» Simple ML
«subsystem» Training data
«subsystem» Training
«subsystem» Deploy
«subsystem» Consumer applcation
«subsystem» Production data

- Training
- Deployment
- Prediction generation
- No structured monitoring

Sufficient for predictions – not sufficient for robustness.

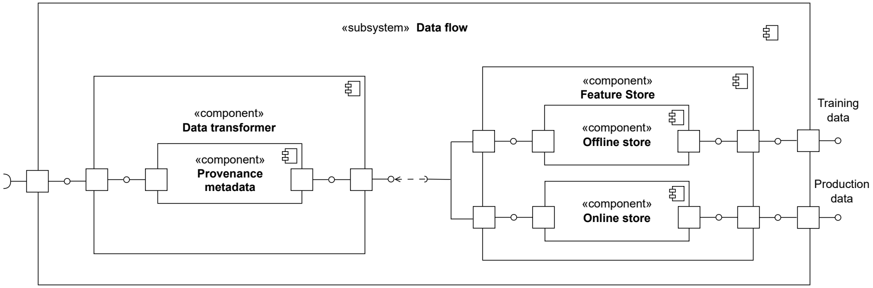
Architecture for Enabling MLOps



- Structured using UML subsystems
- Explicit separation of concerns
- Monitoring as a first-class subsystem
- Supports heterogeneous tools
- Enable feedback loops

Note: This is a conceptual structuring proposal, not a fully implemented platform.

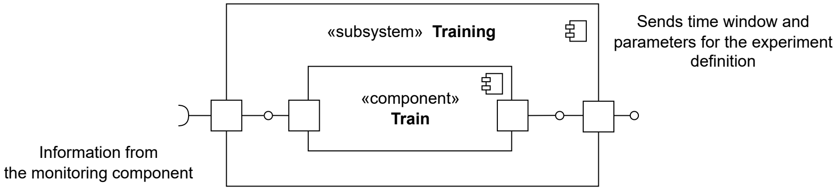
Data Flow Subsystem


«subsystem» Data flow
«component» Data transformer
«component» Provenance metadata
«component» Feature Store
«component» Offline store
«component» Online store
Training data
Production data

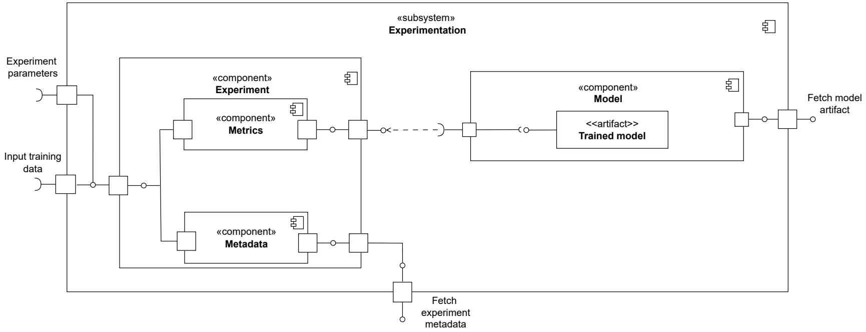
- Data ingestion
- Transformation tracking
- Provenance management
- Feature store integration

Traceable data lineage is fundamental for monitoring.

Training Subsystem


«subsystem» **Training**
«component» **Train**
Sends time window and parameters for the experiment definition
Information from the monitoring component

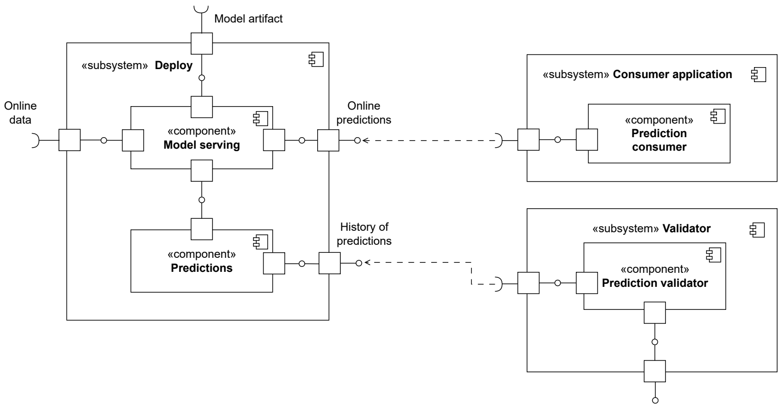
Experimentation Subsystem



Training and Experimentation in the Feedback Loop

- Experiment triggering and parameter selection
- Artifact and metric tracking
- Full traceability of experiments
- Monitoring signals may influence new experiments

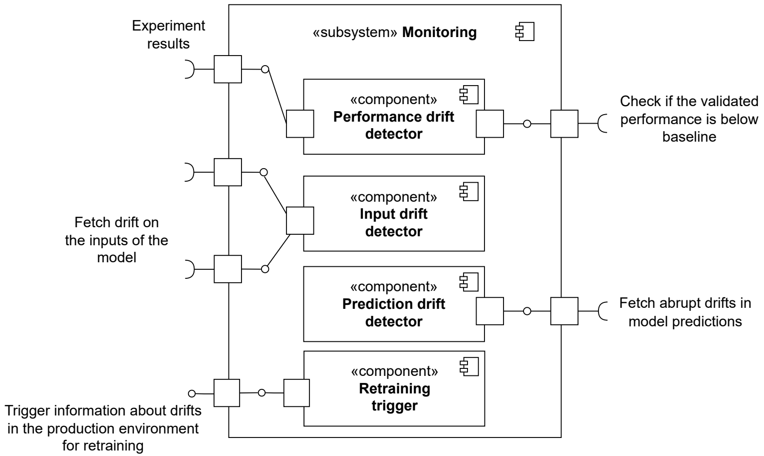
Deployment, Consumption and Validation



- Model serving
- Prediction generation
- External consumption
- Separate validation subsystem

Validation provides structured feedback to earlier stages.

Monitoring as a Central Subsystem

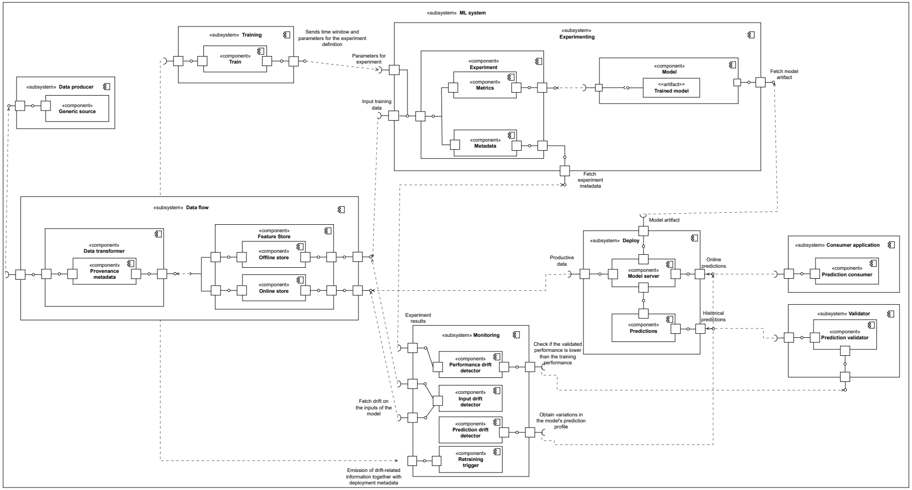

Experiment results
«subsystem» Monitoring
«component» Performance drift detector
Check if the validated performance is below baseline
«component» Input drift detector
Fetch drift on the inputs of the model
«component» Prediction drift detector
Fetch abrupt drifts in model predictions
«component» Retraining trigger
Trigger information about drifts in the production environment for retraining

Monitoring subsystem - Key Aspects

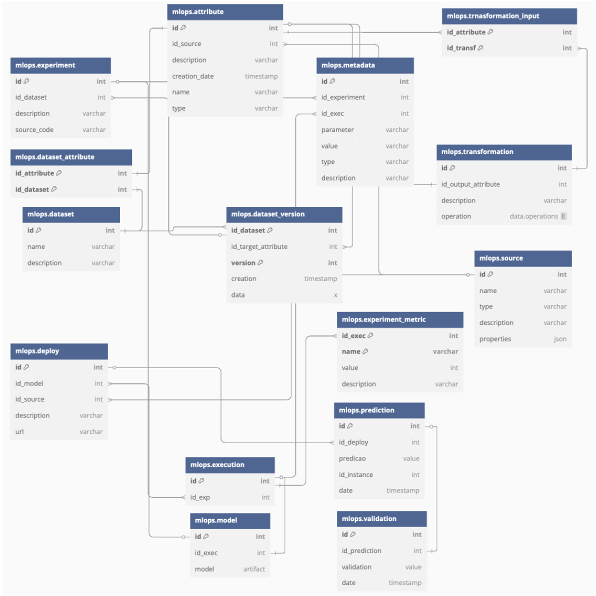
- Input distribution monitoring
- Prediction profile monitoring
- Error-rate validation
- Infrastructure monitoring

Drift detection methods studied in this thesis operate within this subsystem.

Architecture Overview


«subsystem» ML system
«subsystem» Training
«component» Train
Sends time window and parameters for the experiment definition
Parameters for experiment
«subsystem» Experimenting
«component» Experiment
«component» Metrics
«component» Metadata
«component» Model
<<artifact>> Trained model
Fetch model artifact
Input training data
Fetch experiment metadata
«subsystem» Data producer
«component» Generic source
«subsystem» Data flow
«component» Data transformer
«component» Provenance metadata
«component» Feature Store
«component» Offline store
«component» Online store
Model artifact
«subsystem» Deploy
«component» Model server
«component» Predictions
Productive data
Online predictions
Historical predictions
«subsystem» Consumer application
«component» Prediction consumer
«subsystem» Validator
«component» Prediction validator
Experiment results
«subsystem» Monitoring
«component» Performance drift detector
«component» Input drift detector
«component» Prediction drift detector
«component» Retraining trigger
Check if the validated performance is lower than the training performance
Fetch drift on the inputs of the model
Obtain variations in the model's prediction profile
Emission of drift-related information together with deployment metadata

Logical Data Model Supporting the Architecture



Logical Data Model Supporting the Architecture

- Unique identifiers for all artifacts
- Dataset versioning
- Transformation traceability
- Experiment $\rightarrow$ Model $\rightarrow$ Deployment linkage
- Prediction $\rightarrow$ Validation linkage

Enables structured feedback and cross-stage analysis.

From Architecture to Drift Experiments

- Monitoring subsystem includes:
  - Distribution-based drift detection
  - Error-rate monitoring
- Experiments simulate how monitoring would trigger retraining strategies

Although not fully implemented, the experiments instantiate the monitoring logic envisioned in this architecture.

Outline

Drift detection experiment

- Empirically evaluate input data drift detection techniques
- Analyze their behavior under:
  - Concept drift scenarios
  - Controlled data drift scenarios
- Study the influence of batch size

Goal: Understand how monitoring techniques behave before integrating them into ML systems.

Drift Detection Techniques

- **KS95** – Kolmogorov-Smirnov test ($\alpha = 0.05$)
- **KS90** – Kolmogorov-Smirnov test ($\alpha = 0.10$)
- **HDDDM** – Hellinger Distance-based method
- **JSDDM** – Jensen-Shannon Divergence-based method
- **Base** – No drift detection (reference)

Batch sizes evaluated: 1000, 1500, 2000, 2500

Datasets Overview

Concept Drift Datasets

- Insects (multiple drift patterns)
- SEA / MULTISEA
- STAGGER / MULTISTAGGER
- Electricity

Data Drift Dataset

- Magic Gamma Telescope (modified)

Controlled Synthetic Datasets

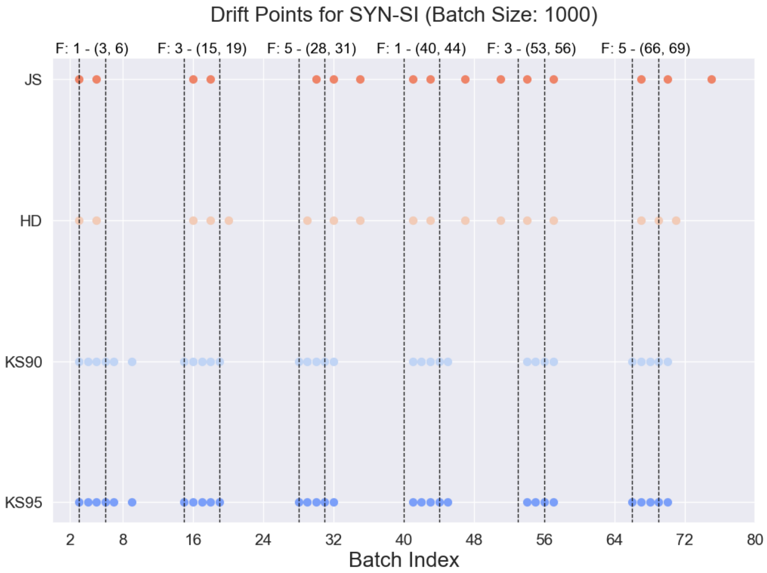
- SYN (no drift)
- SYN-PA, SYN-PI
- SYN-SA, SYN-SI

Controlled Synthetic Drift Design

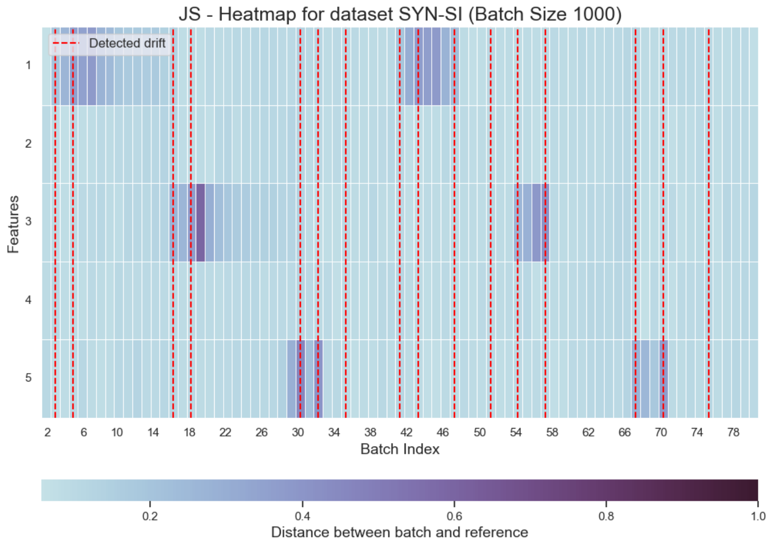
- 80,000 samples
- 5 features (3 drifted, 2 control)
- Two drift types:
  - Abrupt
  - Incremental
- Two drift patterns:
  - Parallel (multiple features drift simultaneously)
  - Switching (sequential feature drift)

Allows precise evaluation of detection accuracy.

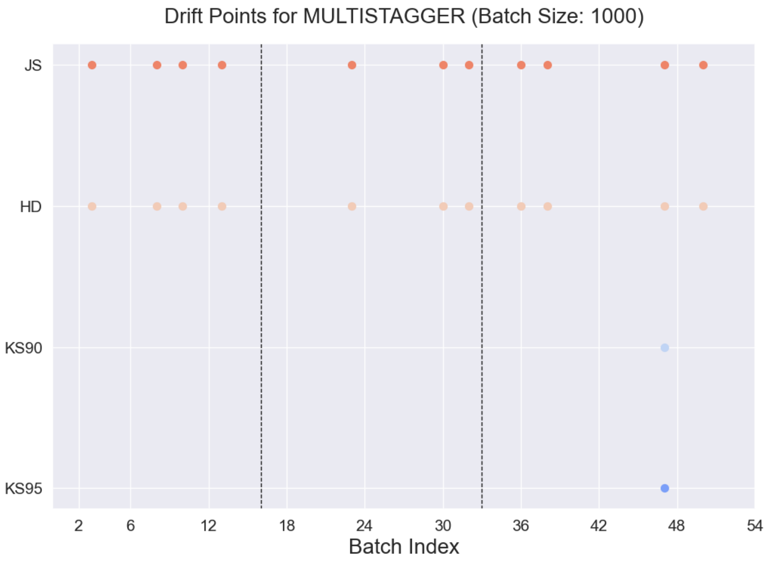
Example: Detected Drifts (Synthetic Data Drift)



Example: JS Heatmap (Synthetic Data Drift)



Example: Detected Drifts (Concept Drift Scenario)

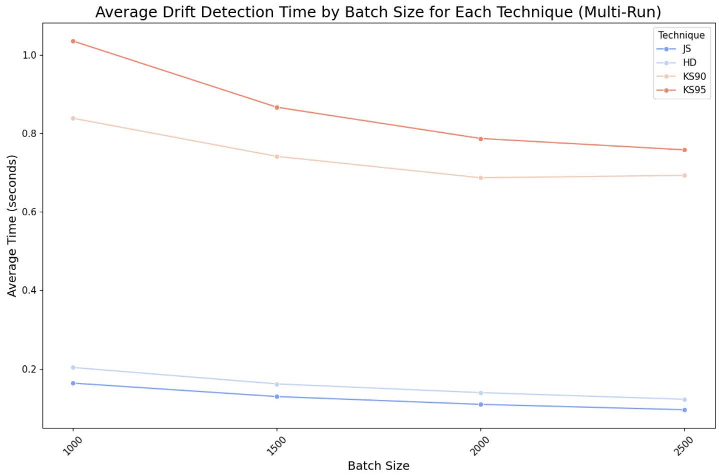


Main Findings (Detection Behavior)

- KS-based methods are more sensitive
- HDDDM and JSDDM are more conservative
- Incremental drifts are detected more consistently
- Larger batches increase recall
- No single technique dominates across all scenarios

Drift detection behavior is context-dependent.

Execution Time vs Batch Size



- Smaller batches $\rightarrow$ more overhead
- Larger batches $\rightarrow$ lower total execution time
- Trade-off between adaptation speed and computational cost

Outline

Using monitored drifts experiment

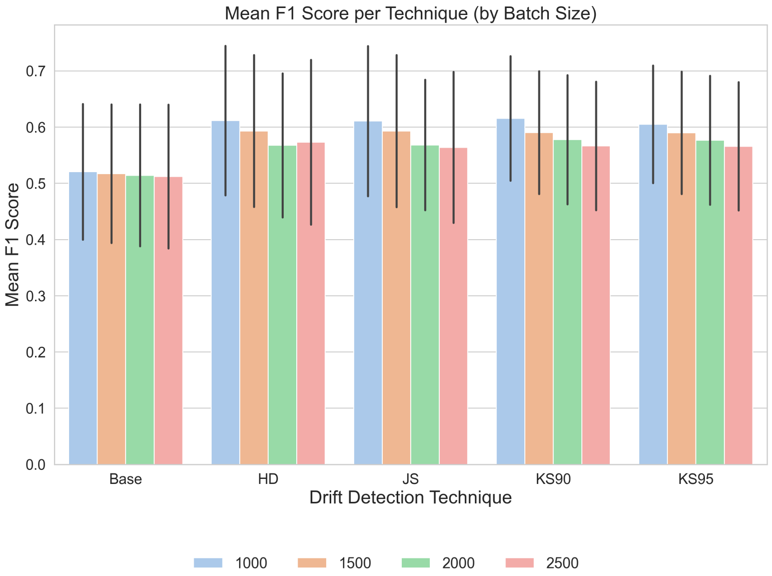
- Evaluate whether drift-aware retraining improves performance
- Compare:
  - Baseline incremental model (C_B)
  - Drift-aware model (C_D)
- Classifier metrics:
  - F1-score
  - AUC

Does monitoring input data improve ML systems?

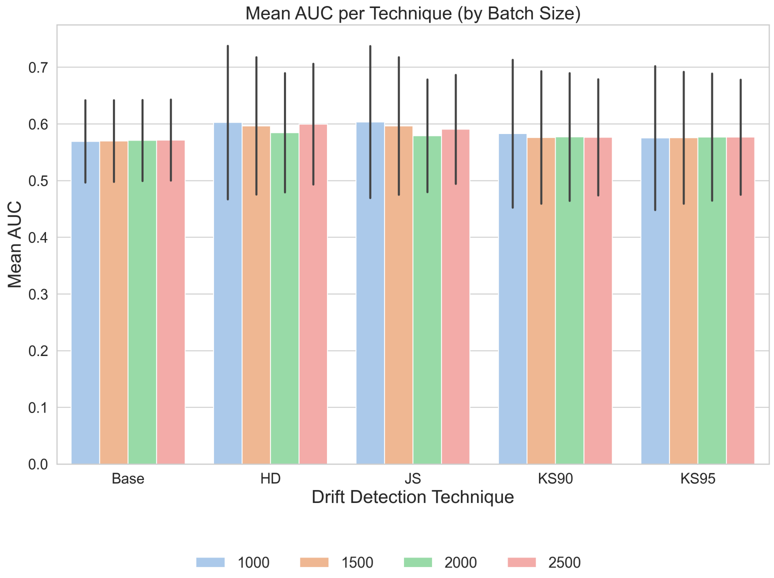
Prequential Evaluation Strategy

- Initialize models using the first batch:
  - Train baseline classifier C_B
  - Train drift-aware classifier C_D
  - Set batch 1 as reference set
- For each new batch t:
  - Generate predictions with C_B and C_D
  - Compute drift statistic $\delta(t)$ between reference and batch t
  - Compare $\delta(t)$ with detection threshold θ
- Model update:
  - C_B: always incrementally updated
  - C_D:
    - no drift $\rightarrow$ update incrementally
    - drift detected $\rightarrow$ reset and retrain from reference batch

Classifier: Performance Impact Across Datasets



Overall Performance Summary



Overall Performance Summary

- All drift-aware approaches outperform baseline in F1
- KS90 achieves highest F1 for small batches
- HD and JS show competitive AUC
- No universally superior method

Results Interpretation

Classifier Performance

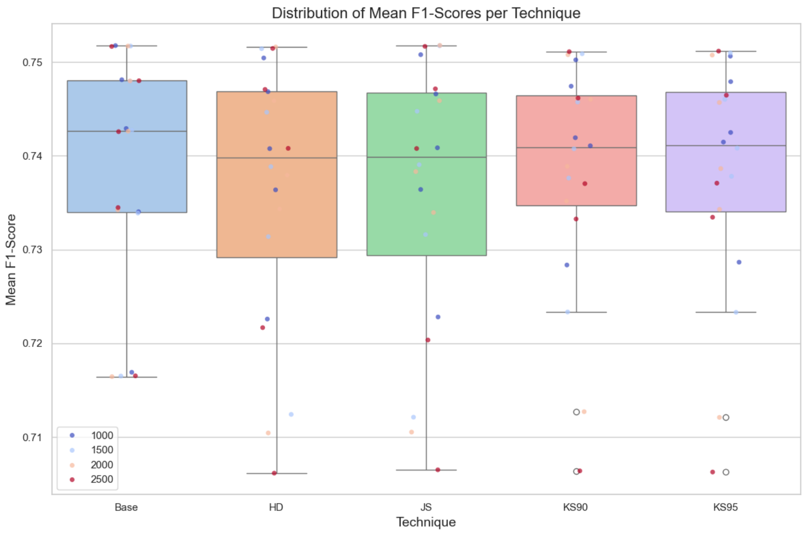
- Drift-aware models often outperform the baseline
- Largest improvements observed in datasets with strong drift
- Examples: *Magic* and SYN-SI

Impact of Batch Size

- Best F1 generally observed with batch size = 1000
- Smaller batches allow faster adaptation
- Larger batches delay detection

Trade-off: adaptation speed vs computational efficiency

Multi-Run Synthetic Experiments



Multi-Run Synthetic Experiments

- KS-based methods show stable performance across runs
- More drift detections do not necessarily imply better F1

Key Takeaways

- Data drift significantly impacts predictive performance
- Monitoring input data improves system robustness
- Detection effectiveness depends on:
    - Drift type
    - Dataset characteristics
    - Batch configuration
- No universally superior drift detection method

Effective monitoring improves ML systems, but the best strategy depends on context.

Outline

Research Focus

This thesis investigated:

- How **data drift** affects machine learning systems
- How different drift detection techniques behave under diverse scenarios
- How **software architecture** can support reliable ML operation

Scientific Dissemination

The results of this research were validated through peer review:

- Helfstein, L., and Braghetto K.R. *An Empirical Analysis of Data Drift Detection Techniques in Machine Learning Systems*. In: **Proceedings of the Brazilian Symposium on Databases (SBBD)**, 2024.
- Helfstein, L., and Braghetto K.R. *Evaluating Data Drift Detection and Its Effects on Machine Learning System Performance*. In: **Journal of Information and Data Management (JIDM)**, 2026.

Contribution 1: Architecture for ML Systems

- Structured lifecycle organization:
  - Data Flow
  - Training Experimentation
  - Deployment
  - Monitoring
  - Feedback Loops
- Emphasizes:
  - Data versioning
  - Reproducibility
  - Monitoring of input distributions
  - Explicit retraining mechanisms

Provides architectural clarity for maintainable ML systems.

Contribution 2: Empirical Analysis of Drift Detection

- Evaluation of statistical and distance-based methods:
  - Jensen–Shannon
  - Kullback–Leibler
  - Kolmogorov–Smirnov
  - Others
- Tested across:
  - Real and synthetic datasets
  - Sudden, gradual, and incremental drifts

Finding: Input monitoring is an effective strategy for early drift identification.

Broader Perspective

- Reliability depends on more than model accuracy
- Engineering practices are essential:
    - Observability
    - Experiment tracking
    - Data lineage
    - Feedback loops

ML systems must be designed to evolve, not only to predict.

Future Work

- Automated and adaptive retraining pipelines
- Benchmark suites for drift detection
- Evaluation with complex ML models
- Practitioner-centered empirical studies

Bridging research and real-world ML engineering remains an open challenge.

Thank you.

References I

Breck, Eric, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley (2017). "The ML test score: A rubric for ML production readiness and technical debt reduction". In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 1123–1132.

Dasu, Tamraparni, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi (2006). "An information-theoretic approach to detecting changes in multi-dimensional data streams". In: *Symposium on the Interface of Statistics, Computing Science, and Applications (Interface)*.

Ditzler, Gregory and Robi Polikar (2011). "Hellinger distance based drift detection for nonstationary environments". In: *2011 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*, pp. 41–48.

References II

- Gama, Joao and Gladys Castillo (2006). “Learning with Local Drift Detection”. In: *Advanced Data Mining and Applications (ADMA 2006)*. Vol. 4093. Lecture Notes in Computer Science. Springer, pp. 42–55. DOI: `10.1007/11811305_4`.
- Kreuzberger, Dominik, Niklas Kühl, and Sebastian Hirschl (2023). “Machine learning operations (mlops): Overview, definition, and architecture”. In: *IEEE access* 11, pp. 31866–31879.
- Lu, Jie, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang (2018). “Learning under concept drift: A review”. In: *IEEE transactions on knowledge and data engineering* 31.12, pp. 2346–2363.
- Rabanser, Stephan, Stephan Günnemann, and Zachary Lipton (2019). “Failing loudly: An empirical study of methods for detecting dataset shift”. In: *Advances in Neural Information Processing Systems* 32.

References III

Sato, Danilo, Arif Wider, and Christoph Windheuser (2019). "Continuous delivery for machine learning". In: *Martin Fowler* 9.

Schröder, Tim and Michael Schulz (2022). "Monitoring machine learning models: a categorization of challenges and methods". In: *Data Science and Management* 5.3, pp. 105–116.

Sculley, D., Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean François Crespo, and Dan Dennison (2015). "Hidden Technical Debt in Machine Learning Systems". In: *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*, pp. 2503–2511. URL: https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.

References IV

Serban, Alex and Joost Visser (2022). “Adapting software architectures to machine learning challenges”. In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, pp. 152–163.