# High-Level Solutioning for Model Monitoring in Banking

## Industry Standards and Best Practices in Financial Services

Financial regulators and industry bodies emphasize robust model monitoring as a core component of Model Risk Management (MRM). The Basel Committee’s framework calls for **ongoing monitoring** programs that include *regular reviews of model performance* and prompt updates to models as needed[[1]](https://empoweredsystems.com/blog/what-is-the-basel-morm-framework-for-model-risk-management/#:~:text=all%20technical%20requirements,analysis%20and%20corrective%20action%20plans). Similarly, the US Federal Reserve’s SR 11-7 guidance and OCC guidelines highlight continuous monitoring to verify models perform as intended under changing conditions[[2]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L543%20Ongoing%20monitoring,It%20is%20important)[[3]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L584%20In%20accordance,It). Weak monitoring has led to severe consequences – e.g. a large bank’s $2B loss in 2012 from an inadequately monitored risk model[[4]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=Weakness%20in%20an%20organization%E2%80%99s%20ongoing,Risk) – underscoring why regulators require proactive surveillance of data quality, model outputs, and concept drift. European regulators share this view: the Bank of England’s 2023 principles note that **performance monitoring becomes increasingly important as AI/ML model complexity grows**[[5]](https://www.bankofengland.co.uk/prudential-regulation/publication/2023/may/model-risk-management-principles-for-banks#:~:text=,%E2%80%93%20such%20ethical%20challenges%20could). Emerging AI governance standards like **NIST’s AI Risk Management Framework** and **ISO/IEC 42001** also stress continuous monitoring and feedback. ISO 42001, for example, explicitly requires organizations to *“monitor and manage changes in AI behavior,”* performing periodic reviews and impact assessments to ensure models remain safe and effective[[6]](https://www.brightdefense.com/resources/iso-42001-compliance/#:~:text=Organizations%20must%20establish%20data%20collection%2C,to%20operate%20safely%20and%20ethically). NIST’s framework likewise includes **“Manage”** functions that implement risk controls and **continuous monitoring** processes to maintain trustworthy AI[[7]](https://www.diligent.com/resources/blog/nist-ai-risk-management-framework#:~:text=monitoring%20processes%20to%20reduce%20identified,roles%20for%20ongoing%20AI%20risk). In practice, **best practices** include:

* **Establish clear governance for monitoring** – define roles (model owners, risk officers) and procedures for reviewing drift reports and escalating issues. Findings from monitoring should be documented, reported to management, and *escalated up to the board* in regulated firms[[8]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L665%20actively%20managed,escalated%20up%20to%20the%20board). This aligns with governance expectations that senior leadership remain informed of model performance and compliance.
* **Monitor both data and outcomes** – Track input data quality (accuracy, completeness) and drifts, as well as model outputs and accuracy against ground truth. SR 11-7 specifically notes verifying that *data inputs remain accurate, complete, and consistent with the model’s purpose*[[3]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L584%20In%20accordance,It) as part of monitoring.
* **Leverage reference baselines** – Use a stable reference dataset (e.g. test set or prior period data where model performed well) rather than the training set alone, to set realistic baseline expectations[[9]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Many%20people%20and%20tools%20advocate,with%20the%20model%E2%80%99s%20test%20set)[[10]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=I%20ran%20some%20experiments%20using,as%20the%20reference%20for%20monitoring). This avoids falsely narrow thresholds from overfit training data. Regulators (e.g. ECB, Basel) expect monitoring against benchmarks and back-testing as evidence of ongoing validation.
* **Periodic revalidation** – Beyond continuous metrics, industry best practice is to schedule deeper periodic validations (annual or quarterly) to re-assess model assumptions, as recommended by frameworks like Basel MRM and ECB’s internal model guidelines. This ensures any concept drift or data changes that accumulate over time are formally addressed[[11]](https://www.bankofengland.co.uk/prudential-regulation/publication/2023/may/model-risk-management-principles-for-banks#:~:text=,the%20lowest%20tier%20models%3B%20and).
* **Bias and fairness checks** – Financial services firms are encouraged (and in some jurisdictions, required) to monitor models for bias and fairness continuously. The NIST AI framework calls for guardrails to detect discriminatory outcomes and bias in models[[12]](https://www.diligent.com/resources/blog/nist-ai-risk-management-framework#:~:text=match%20at%20L427%20or%20transaction,to%20detect%20discriminatory%20outcomes%2C%20monitor), and ISO 42001 includes transparency and fairness in automated decisions[[13]](https://www.brightdefense.com/resources/iso-42001-compliance/#:~:text=%2A%20Automatic%20Decision,ISO%2042001%20mandates%20rigorous%20data). In practice, this means tracking segment-wise performance (e.g. by demographic slices) as part of model monitoring, and alerting if disparities arise.
* **Robust documentation** – Every monitoring metric, drift detection result, and alert resolution should be logged. Regulators expect a documented audit trail of model performance over time, including when alerts were triggered and how the team responded[[8]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L665%20actively%20managed,escalated%20up%20to%20the%20board). This documentation supports compliance audits (e.g. for SR 11-7 or ECB expectations) and internal model governance reviews.

By aligning with these standards – Basel’s sound practices, NIST/ISO frameworks, and specific regulator guidance – banks can design monitoring systems that not only maintain model efficacy but also satisfy **risk and compliance requirements** for transparency, accountability, and timely incident response[[1]](https://empoweredsystems.com/blog/what-is-the-basel-morm-framework-for-model-risk-management/#:~:text=all%20technical%20requirements,analysis%20and%20corrective%20action%20plans)[[14]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=Ongoing%20monitoring%20establishes%20a%20framework,It%20is%20important).

## Architecture Options for Model Monitoring

Modern MLOps offers multiple architectural approaches to implement model monitoring and drift detection. We compare three common patterns – using Databricks integrated dashboards, a centralized Prometheus/Grafana observability stack, and an MLflow-centric solution – highlighting design considerations for each.

### Option 1: Databricks Lakehouse Monitoring and Dashboards

Databricks now provides native **Lakehouse Monitoring** capabilities to track data and model metrics within its platform. This approach attaches monitors to Delta tables (including **inference tables** that log model inputs, predictions, and optionally labels) and automatically computes statistics and drift metrics over time windows[[15]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=Databricks%20Lakehouse%20Monitoring%20lets%20you,data%20quality%20and%20model%20performance)[[16]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=,performing%20better%20than%20version%20B). The results are stored in structured **metrics tables** and visualized through Databricks SQL dashboards. **Figure 1** illustrates how data flows from production tables into the monitoring engine and dashboard:

*Figure 1: Databricks Lakehouse Monitoring tracks the statistical properties of data tables and model inference results. It can detect changes in data distributions or model performance and alert users to drift or quality issues*[*[17]*](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=To%20draw%20useful%20insights%20from,help%20you%20identify%20the%20cause)*. The platform creates profile metrics (e.g. summary stats per feature) and drift metrics (e.g. distribution shift measures) which are queryable and visualizable in dashboards.*

Under this architecture, **drift detection is largely automated**. When a monitor runs, Databricks computes an array of drift metrics comparing each feature’s distribution in the current window vs. a baseline or previous window[[18]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=When%20a%20monitor%20runs%20on,and%20a%20drift%20metrics%20table)[[19]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=the%20metrics%20in%20the%20row,null%20values). For numeric features, metrics include Kolmogorov–Smirnov test statistics, Wasserstein distance, and Population Stability Index, while categorical features get chi-square test, Jensen–Shannon divergence, etc.[[20]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=chi_squared_test%20%60struct,distance%20for%20drift%20in%20distribution)[[21]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=js_distance%20double%20Jensen%E2%80%93Shannon%20distance%20for,for%20categorical%20columns). For example, Lakehouse Monitoring calculates PSI as a numeric indicator of distribution change, with guidelines like *PSI < 0.1 = no significant change; PSI ≥ 0.2 = significant drift*[[22]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=,2%20indicates%20significant%20population%20change). These metrics, along with data quality indicators (missing value %, etc.), are stored in Delta tables that the Databricks SQL dashboard queries and visualizes. Teams can set up alerts on these metrics (e.g. via queries in Databricks SQL or jobs) to notify if drift exceeds thresholds.

**Advantages:**  
- *Integrated and Unified:* All monitoring data resides in the same lakehouse as the model data, simplifying data lineage and access control. Metrics are directly computed on Delta tables using Spark, ensuring scalability on large datasets. The approach supports monitoring both data drift and model performance in one place[[15]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=Databricks%20Lakehouse%20Monitoring%20lets%20you,data%20quality%20and%20model%20performance)[[16]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=,performing%20better%20than%20version%20B).  
- *Rich Metrics Out-of-the-Box:* Databricks provides a **standardized schema** for metrics and a wide range of drift measures by default (e.g. KS test, JS divergence, Wasserstein, PSI)[[21]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=js_distance%20double%20Jensen%E2%80%93Shannon%20distance%20for,for%20categorical%20columns). This reduces the effort to implement statistical tests manually. The metrics schema is structured (with fields for feature name, slice, time window, drift type, etc.), promoting consistency[[23]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=Grouping%20columns%20window%20%60struct,Value%20of%20the%20slicing%20expression)[[24]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=the%20metrics%20in%20the%20row,Number%20of%20columns%20with).  
- *Visualization and Slicing:* Databricks SQL dashboards allow interactive exploration of drift over time and across feature slices. Users can drill down into specific features or segments (e.g. by customer region) to diagnose drift causes. The platform also supports custom slicing logic (the monitor can compute metrics per data slice, such as by category or numerical range)[[25]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=%60slicing_exprs%3D%5B)[[26]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=Additional%20statistics%20are%20calculated%20for,analysis).  
- *Alerts and Automated Monitoring:* Because drift metrics are stored as tables, it’s straightforward to write scheduled queries to check for anomalies and trigger alerts. The Lakehouse Monitoring system can capture and alert on distribution changes or model performance drops as they occur[[17]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=To%20draw%20useful%20insights%20from,help%20you%20identify%20the%20cause). This helps meet the *timeliness* expectation of regulators – issues can be caught and addressed promptly.

**Challenges:**  
- *Platform Lock-In:* This solution works best if your ML pipeline and data are already in Databricks. It is a proprietary ecosystem; integrating external services or data sources requires ingestion into Delta tables. Organizations not using Databricks may find it impractical to adopt solely for monitoring.  
- *Limited External Integrations:* Alerts and dashboards are within Databricks – integrating with enterprise incident management (e.g. centralized ops dashboards or sending notifications to third-party systems) may need extra development. Unlike dedicated observability tools, Databricks doesn’t natively push metrics to systems like PagerDuty or email; users must set up jobs or API calls for alerting.  
- *Real-Time Monitoring Gaps:* The Lakehouse monitoring jobs typically run on a schedule (and current limitations only compute windows up to the last 30 days by default[[27]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=,contact%20your%20Databricks%20account%20team)). This may not be true real-time streaming detection. If sub-daily or streaming monitoring is needed, additional custom streaming jobs or integration with Spark Structured Streaming might be required.  
- *Expertise and Cost:* Utilizing this architecture presumes familiarity with Databricks SQL and possibly incurs compute cost for continuous metric computation. Also, while Databricks automates metric calculations, data scientists must still decide on threshold logic (e.g. what PSI value triggers retraining) and avoid false positives.

**Use Case Fit:** This approach is well-suited for teams already using Databricks for model training or scoring. It offers a **turn-key solution** for **batch or periodic monitoring** of data and models in one environment. Banks leveraging a Lakehouse architecture can embed monitoring into their existing workflows seamlessly. For example, a credit risk model’s input table can be monitored for drift in borrower demographics while its output (credit scores) is monitored for performance divergence – all within Databricks, satisfying internal model risk guidelines with minimal integration overhead.

### Option 2: Central Observability Platform (Prometheus + Grafana)

A second architecture treats model metrics like any other application telemetry, using **Prometheus for metrics collection and Grafana for dashboards/alerts**. In this design, your model serving infrastructure (or batch jobs) exports monitoring data that Prometheus scrapes, and Grafana visualizes these time-series metrics. This approach leverages proven, general-purpose observability tools to monitor ML-specific indicators alongside system metrics.

One proven pattern is to instrument model servers to log **feature distribution histograms** and output distributions. For example, BasisAI’s open-source **Boxkite** library hooks into Python model serving code to collect real-time histograms of model inputs and outputs[[28]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Boxkite%2C%20a%20simple%2C%20open%20source,distribution%20between%20the%20two%20environments)[[29]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Under%20the%20hood%2C%20Boxkite%20leverages,for%20Prometheus%20to%20periodically%20scrape). These histograms (for each feature and prediction) are exposed as Prometheus metrics, which Prometheus scrapes on a schedule[[30]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=match%20at%20L547%20Under%20the,for%20Prometheus%20to%20periodically%20scrape). Grafana can then plot the production feature distributions against training or baseline distributions to detect drift. **Figure 2** shows an overview of this architecture, where the model service provides metrics to Prometheus and Grafana monitors for drift:

flowchart LR  
 subgraph Model Service  
 A[[Model Serving API]]   
 A -- logs predictions--> P[Prediction Log / DB]  
 A -- exports histograms--> M((Prometheus))  
 end  
 subgraph Monitoring  
 M==> Grafana[Grafana Dashboards]  
 Grafana -->|Alerts| OpsTeam[On-call / Alerts]  
 end  
 classDef gray fill:#eee,stroke:#333,stroke-width:1;  
 P:::gray

*Figure 2: Central observability monitoring – the model serving system is instrumented (e.g. via Boxkite) to expose metrics, such as feature value histograms and prediction counts.* *Prometheus* *scrapes these metrics and* *Grafana* *dashboards visualize drift (e.g. overlaying production vs. baseline distributions)*[*[31]*](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=After%20setting%20up%20Prometheus%20as,propagated%20to%20the%20query%20string)[*[32]*](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Grafana%E2%80%99s%20histogram%20visualization%20helps%20us,hand%2C%20shows%20no%20visible%20difference)*. Alerts can be set using PromQL queries, for example triggering if drift metrics like KL divergence or K-S test statistic exceed a threshold.*

With this architecture, you often compute drift metrics in the monitoring layer. In a real deployment, one might maintain training baseline histograms as constants in Prometheus, and as new data histograms come in, use PromQL queries to calculate drift metrics. BasisAI’s team, for instance, uses **Kullback-Leibler divergence** for categorical features and **Kolmogorov–Smirnov (K-S) test** for continuous features, all implemented via PromQL on the collected histograms[[33]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=other%20hand%2C%20shows%20no%20visible,difference). They configure Grafana alerts on those metrics to flag significant drift. This approach effectively unifies ML monitoring with traditional infra monitoring: the same Grafana can show model drift alongside CPU usage, and alerts flow through the same channels.

**Advantages:**  
- *Unified Infrastructure:* Uses battle-tested, open-source tools. By *“taking full advantage of the Prometheus–Grafana ecosystem, [this approach] unifies ML monitoring and software monitoring in a single stack”*[[34]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Conclusion). Operations teams can leverage familiar tooling for ML, reducing the learning curve.  
- *Real-Time Capabilities:* Prometheus is designed for real-time or near-real-time metric scraping (e.g. 15-second to 1-minute intervals). This enables low-latency detection of issues. For online models, one can observe distribution shifts almost immediately after they occur, critical for catching fast-moving concept drift (e.g. fraud patterns).  
- *Flexibility:* Highly customizable – teams define which metrics to collect and alert on. Any statistic expressible in PromQL can be computed. For example, you might track the **rate of predictions in each score band**, the fraction of null inputs, or custom drift metrics. You can incorporate domain-specific rules (like alert if a feature mean shifts by 3σ). This flexibility extends to **dashboard design** – Grafana can combine plots, tables, and even link to external notebooks or analyses.  
- *Separation of Concerns:* The model service emits raw metrics (e.g. histograms or counts), and the monitoring system handles analysis. This means instrumentation can be relatively light on the application, and changes in monitoring logic (e.g. trying a different drift metric) can often be done in the Prometheus/Grafana layer without modifying production code.  
- *Scalable and Language Agnostic:* Prometheus clients exist for many languages, and Grafana supports many data sources. Whether your model is deployed on Kubernetes, a VM, or a cloud function, you can likely integrate it with Prom/Grafana. This makes it suitable for heterogeneous environments and avoids locking into a single ML platform.

**Challenges:**  
- *Setup and Maintenance Complexity:* Unlike an off-the-shelf platform, you must build and maintain the monitoring logic. This involves deploying Prometheus, managing its storage (metrics retention, scaling), and securing it (especially important in finance). Grafana dashboards and alert rules must be created and tuned. This requires DevOps expertise and ongoing maintenance – e.g. ensuring Prometheus doesn’t overload with high-cardinality metrics from ML.  
- *Metrics Volume:* Logging detailed distribution info (histograms for many features, per model version, per data slice) can explode the number of metrics. Careful design is needed to avoid performance issues in Prometheus. Techniques include limiting histogram buckets, exporting summary stats instead, or aggregating client-side. There is a trade-off between granularity and system load.  
- *Statistical Calibration:* Setting alert thresholds on drift metrics can be non-trivial. For example, how high should KL divergence be before raising an alert? There’s a risk of false alarms if thresholds are too sensitive, or missed detections if too lax. Teams often need to experiment and incorporate statistical significance (e.g. only alert on K-S p-value below 0.01). Unlike specialized ML monitoring tools, Prometheus/Grafana don’t natively account for statistical uncertainty – the onus is on the implementers to configure robust detection.  
- *Limited ML Specific Features:* General observability tools may lack some ML-specific functionality. For instance, **concept drift** detection might require comparing model predictions to eventually received ground truth – this joining of data isn’t straightforward in Prometheus (which is focused on numeric time-series). Often, teams supplement this by sending data to a warehouse for offline analysis[[35]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=correct%20predictions%20that%20the%20model,is%20generating%20begins%20to%20degrade), or by writing custom batch jobs that compute performance metrics and push them as Prometheus metrics. Essentially, the out-of-the-box solution covers data drift well; monitoring actual model accuracy requires additional processes.  
- *Data Governance and Security:* Exporting production data distributions to a monitoring system requires careful governance. In finance, one must ensure no sensitive customer data is exposed via metrics. Though Prometheus endpoints can be secured, it’s essential to export *aggregated* stats (histograms, counts) rather than raw data to comply with privacy. Role-based access control to Grafana dashboards is another consideration – who is allowed to see model performance metrics? These concerns mean additional security reviews and potential integration with bank’s IAM systems.

**Use Case Fit:** The Prom+Grafana approach is ideal if you already have a strong observability platform and want to extend it to ML. Many banks have enterprise monitoring for IT operations; integrating model metrics here provides a single pane of glass for reliability and model risk. It’s also well-suited for **real-time scoring systems** (fraud detection, trading models, etc.) where immediate drift detection is needed. For example, a fintech might push model prediction distributions to Prometheus and set Grafana alerts for any sudden distribution shift (potential data pipeline issue or concept drift). This leverages existing ops workflows – e.g., the NOC (Network Operations Center) can respond 24/7 to model alerts just like they do for server downtime. However, for organizations without existing Prom/Grafana expertise, the initial overhead can be significant, and a managed or specialized solution might be preferable.

### Option 3: MLflow-Centric Monitoring

A third approach uses **MLflow**, primarily known for experiment tracking, as the backbone for monitoring model performance and drift. In this architecture, the idea is to log production metrics and drift analysis results as MLflow runs or model versions, leveraging MLflow’s tracking UI and model registry for visualization and comparison. Essentially, MLflow’s tracking server becomes a repository of time-stamped model metrics, and data scientists can analyze trends through the MLflow UI or exported data.

One simple implementation is to create a scheduled job (e.g., a nightly batch) that: fetches the latest production data and model predictions, computes drift metrics and performance metrics (possibly using libraries like Evidently or NannyML), and logs those metrics to MLflow. Each run could be tagged with a date and model version. Over time, MLflow will show a history of, say, model AUC, accuracy, PSI, etc., for each execution. Teams can compare these metrics across time or across model versions in the MLflow UI. If **MLflow Model Registry** is used, you could also update the registered model’s description or metadata with the latest performance info for quick reference.

**Advantages:**  
- *ML Lifecycle Integration:* MLflow is already used in many organizations for experiment tracking and model registry. Extending it to monitoring means **one less tool** to manage. Model training, deployment, and monitoring artifacts can all live in MLflow, creating a continuous record from development to production. For example, a model version in the registry can have associated metrics from training (accuracy on test) and ongoing metrics from production (accuracy on recent data), providing a holistic view.  
- *Experimentation with Monitoring Metrics:* Logging metrics to MLflow makes it easy for data scientists to use Python notebooks to compute custom drift metrics or charts and store them as artifacts. Evidently AI’s reports, for instance, can be generated in a notebook and logged to MLflow as an HTML artifact for each week’s batch[[36]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=You%20can%20go%20pretty%20far,is%20detected%20in%20your%20dataset). This can be an **agile way to evolve monitoring** – you can iterate on what you log without deploying new infrastructure.  
- *Alignment with Model Versioning:* Since MLflow is model-centric, it naturally encourages tracking metrics **per model version**. In regulated industries, it’s important to know which model version is performing better or worse. MLflow can log, say, *Model v1 – January data AUC 0.80; Model v2 – January data AUC 0.82*, making it straightforward to justify why a new model is promoted (better performance) or to detect if an update performs worse than its predecessor.  
- *Open Source and Portable:* MLflow is open-source and can be hosted on-premises, which may satisfy banks that have strict data control requirements. All monitoring data stays in the MLflow backend database/files, which can be in the bank’s controlled environment. Additionally, the MLflow API makes it easy to integrate with other systems or notebooks for custom analysis (e.g., querying the tracking server for the last 6 months of metrics to feed into a report or another visualization tool).

**Challenges:**  
- *No Built-in Alerting:* MLflow by itself does not provide alerting or real-time dashboards. It’s largely a passive store of metrics. To get alerts, you would need to build a separate process that queries MLflow’s metrics and then triggers notifications (or integrate MLflow with Prometheus as a data source, etc.). This means additional engineering if immediate notifications of drift are required. In practice, MLflow-centric monitoring tends to be more *batch and analysis-oriented* rather than instant alerting.  
- *Limited UI for Monitoring Trends:* The MLflow UI shows metrics per run, and you can compare runs, but it’s not a full monitoring dashboard with time-series plots (it’s more geared toward comparing static experiments). Navigating dozens of nightly runs to spot a trend can be cumbersome. You may need to pull the data out and use another tool for a nice time-series visualization. As Evidently’s team notes, as you scale up reports “there is no easy way to track trends” in pure report files[[37]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=Image%3A%20Evidently%20Reports), so a live dashboard might still be needed on top of MLflow for long-term monitoring.  
- *Authentication and Multi-Tenancy:* **User management** in MLflow (open source) is rudimentary – it doesn’t have robust access controls out-of-the-box[[38]](https://blog.devgenius.io/mlflow-for-model-monitoring-cb8b2177b67a?gi=8792005f53bb#:~:text=Conclusion). In a bank, different teams or roles might need restricted access to certain model data. Ensuring only authorized personnel can view certain models’ monitoring info might require running separate MLflow instances or using a commercial version. This is noted as a common pain point (managing different user groups’ access)[[38]](https://blog.devgenius.io/mlflow-for-model-monitoring-cb8b2177b67a?gi=8792005f53bb#:~:text=Conclusion).  
- *Data Volume and Retention:* MLflow tracking is not optimized for high-frequency logging. If you attempted to log every prediction or very granular metrics, MLflow’s backend (often a SQL database or file store) could become a bottleneck. It’s better suited for aggregate metrics computed periodically. The **cadence** of logging might be daily or weekly, which could miss faster issues. Also, unless purged, the tracking store will keep growing; governance of how long to keep historical monitoring runs needs consideration (especially if the metrics include any sensitive info).  
- *Manual Effort & Reliability:* A custom MLflow monitoring pipeline is code that must be written and maintained. If the monitoring job fails, it might go unnoticed (since MLflow won’t alarm on a missing run). Ensuring high availability and reliability of this pipeline is on the implementers. In contrast, dedicated monitoring platforms are built to be always-on and have self-monitoring.

**Use Case Fit:** An MLflow-centric approach can work well for **batch prediction environments or low-frequency model scoring**, where immediate alerts are less critical than thorough periodic analysis. For example, a bank that generates credit risk scores monthly might use MLflow to log each month’s model performance (Gini coefficient, etc.) and drift (PSI on key features) as an MLflow run. The risk team can then review these logs in a monthly model performance meeting, all within the MLflow UI or exported reports – this satisfies regulatory monitoring expectations in a documented way. It’s also a good interim solution for teams who already have MLflow and want to kick-start monitoring **quickly with minimal new infrastructure**. Over time, if needs outgrow MLflow’s capabilities (e.g. needing real-time alerts or more scalable storage), the team might transition to another solution, but MLflow can provide a solid starting point to capture monitoring data in the model’s “system of record.”

## Key Metrics and Techniques for Data Drift and Concept Drift

Effective model monitoring hinges on selecting the right metrics to quantify drift and performance degradation. In financial services, certain metrics have become **standard for detecting drift** due to their interpretability and acceptance by risk managers (and even auditors). We outline key metrics and methods for **data drift** (changes in input data distribution) and **concept drift** (changes in the relationship between inputs and outcomes or changes in model performance).

### Data Drift Detection Metrics

*Data drift* refers to changes in the distribution of model input features or independent variables over time[[39]](https://nannyml.readthedocs.io/en/main/tutorials/detecting_data_drift.html#:~:text=The%20model%20has%20been%20trained,X). In banking, data drift can occur due to evolving customer behavior, economic shifts, data collection changes, etc., and may lead to model mismatch if not caught. Common metrics and techniques include:

* **Population Stability Index (PSI):** Widely used in finance, PSI measures the shift in a variable’s distribution between a baseline (expected) and current sample[[40]](https://arize.com/model-drift/#:~:text=1,Expected). It buckets values and sums the differences; as a rule of thumb, PSI >= 0.2 indicates significant change[[22]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=,2%20indicates%20significant%20population%20change). PSI is popular because it provides a single number per feature that risk teams can track monthly; it’s used in credit scoring model monitoring to track population changes. Regulators are familiar with PSI in model documentation, making it a handy choice for reports.
* **Statistical Distance Metrics:** These compare full distributions:
* *Wasserstein-1 Distance (Earth Mover’s Distance):* Measures how far the distribution has “moved” by computing the minimal transport cost to turn the baseline distribution into the current distribution[[41]](https://arize.com/model-drift/#:~:text=these%20cause%20issues%20with%20KL,euclidean%20distance%20check%20determines%20if). Useful for numeric features (included in Databricks’ drift metrics[[21]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=js_distance%20double%20Jensen%E2%80%93Shannon%20distance%20for,for%20categorical%20columns)) and has intuitive units (same scale as the variable).
* *Kullback–Leibler (KL) Divergence / Jensen–Shannon Divergence:* Measure how one probability distribution diverges from another. KL and its symmetric variant JS are effective for categorical distributions or binned numeric data[[42]](https://arize.com/model-drift/#:~:text=monitor%20and%20troubleshoot%20drift%2C%20including,EMD). For example, if the distribution of transaction types shifts, KL divergence will quantify that change. Many monitoring tools (Arize, Evidently) include these distances[[42]](https://arize.com/model-drift/#:~:text=monitor%20and%20troubleshoot%20drift%2C%20including,EMD), and they are useful to flag which feature’s distribution has changed the most.
* *Kolmogorov–Smirnov (K-S) Test:* A non-parametric test that gives a statistic (D) and p-value for the hypothesis that two samples come from the same distribution. Common in banking for validating scorecard distributions, K-S is effective for continuous data drift detection[[33]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=other%20hand%2C%20shows%20no%20visible,difference). A low p-value indicates drift with statistical significance. K-S is used in Baseline vs. Latest comparisons to add rigor – for instance, an alert might require both a high distance (magnitude) and a significant p-value (to avoid noise).
* *Chi-square Test:* For categorical features, a chi-square test can detect if the frequency distribution across categories has changed beyond random chance[[43]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=non_null_columns_delta%20%60struct,for%20categorical%20columns). If, say, a loan application model suddenly sees a spike in applications from a certain industry category, chi-square would flag the distribution change.
* **Univariate vs. Multivariate Methods:** Univariate metrics examine one feature at a time. They are easy to compute and attribute (you know which feature drifted). However, they can miss *joint distribution* shifts[[44]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=Yet%2C%20due%20to%20the%20multidimensional,Guide%20to%20the%20Multivariate%20Approach) – e.g., if two features covary in a new way but marginal distributions of each haven’t changed, univariate metrics won’t catch it. To address this:
* *PCA-based Drift (Multivariate Reconstruction):* Techniques like PCA reconstruction error compress the feature space and detect drift in the joint space[[45]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=Multivariate)[[46]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=Data%20Reconstruction%20with%20PCA). If new data cannot be well reconstructed by PCA built on reference data, it signals a shift in the correlation structure.
* *Domain Classifier (Classifier Drift):* Train a classifier to distinguish between reference data and current data[[45]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=Multivariate). If the classifier can easily tell which period a sample comes from (e.g., high AUC), it means the joint distribution has shifted significantly[[47]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=The%20differences%20between%20these%20two,detection%20mechanisms%20are%20summarized%20below)[[48]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=NannyML%20has%20two%20methods%20to,in%20this%20other%20%2022). This is a powerful technique; for example, some banks use it on high-dimensional transaction data to detect subtle changes in customer behavior patterns that univariate metrics miss.  
  Multivariate methods are more computationally intensive and harder to explain to non-technical stakeholders, so in practice they complement, rather than replace, simpler metrics. A best practice is to apply multivariate drift detection for high-impact models or when univariate metrics fail to explain performance drops[[49]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Red%20flag%204%3A%20focusing%20only,and%20not%20on%20model%20performance)[[50]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=1,shift%20occurs%20in%20similar%20regions).
* **Data Quality Indicators:** Often overlooked in “drift” discussions, basic data quality metrics are crucial. Monitoring the rate of missing values, zeros, outliers, or new categorical levels can preempt drift. For instance, a spike in null rates in a feature might indicate an upstream data issue which is a precursor to model drift. Banks set thresholds for these (e.g., if missing >5% in a month, investigate). These can be collected as profile metrics (Databricks does this with percent\_null, distinct\_count, etc.[[18]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=When%20a%20monitor%20runs%20on,and%20a%20drift%20metrics%20table)[[51]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=comparison%20window,square%20test%20for)). While not drift per se, they ensure the input data remains within the expected bounds – a requirement in many model monitoring policies.
* **Visualization and Manual Review:** Beyond numeric metrics, generating plots is valuable for conceptually understanding drift. Tools like **Evidently** can produce data drift reports with histograms side-by-side[[52]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=Once%20you%20run%20the%20code%2C,it%20up%20in%20a%20report). Often, model validators or business users will inspect these visualizations monthly. For example, seeing the distribution of loan amount this quarter vs last quarter can give intuitive insight (maybe an influx of small loan applications) that complements the numeric metrics.

In financial contexts, combining these metrics provides a robust picture. A practical approach: use PSI as a broad screening (for each feature each period), then investigate features with high PSI using tests like K-S or plots to confirm and assess impact. Many open-source libraries (Evidently, NannyML, Deepchecks) implement these metrics, and commercial tools (Arize, Fiddler, WhyLabs) incorporate them as well[[42]](https://arize.com/model-drift/#:~:text=monitor%20and%20troubleshoot%20drift%2C%20including,EMD). It’s wise to align on a *metric schema* – e.g., decide that for each feature you will log count, mean, std, PSI, and K-S p-value. Standardizing this makes it easier to report to risk committees and regulators. For instance, one could adopt the schema similar to Databricks’ (profile table + drift table structure)[[18]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=When%20a%20monitor%20runs%20on,and%20a%20drift%20metrics%20table)[[53]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=Column%20name%20Type%20Description%20Grouping,7) to ensure consistency in data collection.

### Concept Drift and Performance Metrics

*Concept drift* refers to changes in the underlying relationship between inputs and the target outcome[[54]](https://nannyml.readthedocs.io/en/main/tutorials/detecting_data_drift.html#:~:text=Data%20drift%20is%20one%20of,drift%20and%20concept%20drift%20simultaneously) – effectively, the model’s concept of the mapping has shifted. In a supervised learning context, this often manifests as model performance degradation: the model’s predictions become less accurate because the true data generating process changed. For example, a fraud detection model may experience concept drift if fraudsters change tactics (the same inputs X now correspond to a different likelihood of fraud Y than before).

Key techniques and metrics to monitor concept drift include:

* **Model Performance KPIs:** The most direct way to detect concept drift is via *realized performance metrics* (when ground truth is available). These include:
* *Accuracy, Precision/Recall, F1-Score, AUC (for classification), RMSE/MAE (for regression), etc.* – calculated on recent actuals vs predictions. For instance, a drop in AUC from 0.85 to 0.75 over two months for a credit default model is a red flag for concept drift. In regulated industries, specific metrics like **Gini coefficient** (related to AUC) are tracked for credit risk models, and minimum thresholds might be set before a model must be reviewed or retrained. Monitoring such KPIs over time is essential – ideally plotted as a time series.
* *Population Stability of Outcomes:* Not only features, but the *target variable distribution* can drift (often called **label shift**). For example, if the default rate in the portfolio goes up due to economic downturn, that’s target drift – it might hurt model calibration. Monitoring the base rate (prevalence of class 1 vs 0) via something like PSI or just percentage change is useful. Significant changes in outcome rates can either indicate concept drift or external changes that the model might need to adapt to.
* *Segment-wise Performance:* Checking performance on important sub-populations (e.g., by region, product type, demographic) can uncover *localized concept drift*. Perhaps overall accuracy is fine, but for young customers it dropped sharply – this could signal concept drift in that segment’s behavior. Regulators encourage such analysis to ensure models remain consistently performant and fair. Tools (including MLflow and Arize) allow logging metrics by segment for this reason[[55]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=We%20use%20two%20metrics%20to,explained%20in%20our%20demo%20video).
* **Estimated Performance (when actuals are delayed or unavailable):** In many financial use cases, you don’t get immediate ground truth. E.g., credit default might only be known after 12 months, or a model that flags suspicious transactions may only be confirmed by investigation weeks later. Waiting for outcomes to measure performance drift is too slow. Techniques have emerged to **estimate performance without immediate labels**:
* *Proxy Metrics:* Use surrogate signals. For instance, a credit model might monitor the repayment status after 30 days as a proxy for eventual default. Not perfect, but if significantly more loans are overdue at 30 days than historically, it could indicate a drift in risk not yet seen in full default rate.
* *Confidence Drift:* If a model is probabilistic, monitor distribution of prediction confidences. A concept drift might cause the model to become less certain (predicted probabilities cluster toward 0.5) or overly certain but wrong. A rising entropy of the prediction distribution or an increase in the frequency of low-confidence predictions can hint that the model is seeing strange new patterns[[56]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=Simple%20strategies%20such%20as%20monitoring,cost%20way%20to%20start%20monitoring).
* *Unsupervised Performance Estimation:* Methods like **DLE (Drift-Driven Loss Estimation)** from NannyML train a secondary model to predict the main model’s error based on inputs[[57]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=I%20used%20one%20of%20NannyML%E2%80%99s,my%20surprise%2C%20there%20was%20none)[[58]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Under%20the%20hood%2C%20DLE%20trains,provides%20us%20with%20a%20way). Essentially, it uses feature drift signals to infer if performance likely dropped. In NannyML’s case, their example showed estimated RMSE tracking actual RMSE closely during a drift, even without having the ground truth at the time[[59]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=none). This approach is complex but powerful for **early warning** of concept drift, especially in scenarios like banking where outcomes can take a long time to observe.
* *Champion-Challenger Models:* Sometimes firms keep an older model or an alternative model running in parallel (shadow mode) and compare outputs. If the new production model starts disagreeing heavily with the challenger or with human expert rules, it may indicate drift. This isn’t a metric per se, but a strategy to catch concept drift via discrepancies.
* **Drift vs. Performance Analysis:** It’s important to note, *data drift does not always lead to performance drop*. As one study put it: *“Not every drift affects model performance”*[[60]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=This%21%20This%20is%20a%20really,Performance)[[50]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=1,shift%20occurs%20in%20similar%20regions). Sometimes inputs shift in distribution but in benign ways that the model can handle; conversely, performance can degrade even with no obvious input distribution changes (perhaps the relationship changed). Best practice is to monitor **both** data drift metrics and direct performance metrics. An observed drift should prompt checking performance, and a performance drop should prompt checking for drift in each feature or emerging patterns. By correlating the two, you can prioritize issues: e.g., if data drift is detected but performance is stable, you might watch closely but not retrain immediately[[61]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Most%20people%20and%20tools%20assume,drift%20can%20be%20more%20important). If performance drops without obvious data drift, it might be concept drift due to something like a changing pattern that univariate metrics can’t catch (here, maybe a multivariate analysis or business context analysis is needed).
* **Alerting and Thresholds for Concept Drift:** In practice, organizations set thresholds like: *“If monthly AUC drops below X, trigger retraining.”* The threshold might be based on validation results or business tolerance. For example, a fraud model might require investigation if precision falls by more than 5 points because that could mean more fraud slipping through. Some teams use statistical tests on performance metrics (e.g., compare confusion matrices over time with McNemar’s test for significant change). But often simpler rules suffice given performance metrics can be noisy; combining a short-term drop with a sustained trend can avoid false alarms. Regulators will look for evidence that model owners have predefined performance criteria that, when breached, lead to action (be it retraining, redevelopment, or increased oversight).

In summary, concept drift monitoring is about **closing the feedback loop**: feeding back actual outcomes to verify the model is still accurate. For a highly regulated model, one might have a schedule (say quarterly) where the model is backtested on accumulated actuals – akin to how market risk models are backtested daily in trading (comparing predictions vs actual losses). For credit models, annual validation is common, but continuous monitoring of interim performance is expected. Modern ML monitoring solutions (like Arize or Fiddler) allow tracking actuals and will highlight performance degradation trends automatically, often combining it with data drift views.

One should also integrate **business metrics** into concept drift monitoring. For example, monitor the **approval rate** of loans or the total $$ credit approved by the model each month. If those change drastically, it might indicate the model’s thresholding or scoring distribution shifted (which could be due to drift). These metrics translate model behavior into business impact directly and can sometimes catch issues earlier (e.g., “we’re suddenly approving 5% more loans than usual – is it because the input population changed or the model is less strict?”).

**Techniques Summary:** To detect concept drift: use realized performance where possible, estimate when not, and always contextualize with data drift. Use metrics like AUC, F1 for supervised performance; use error estimation or proxy signals when labels are delayed; and track business KPIs as an extra layer. This layered approach ensures you don’t rely on any single signal. It aligns with the emerging best practice of *monitoring the entire ML pipeline* – inputs (data drift), outputs (predictions drift), and outcomes (performance drift)[[62]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=thresholds%20is%20a%20great%20way,Simple)[[63]](https://snorkel.ai/blog/how-to-apply-and-use-machine-learning-observability/#:~:text=Arize%20AI%20on%20How%20to,data%20drift%2C%20and%20concept%20drift). By doing so, banks can ensure models remain *valid and compliant* with their intended use, as required by model governance policies.

## Operational Considerations: Metrics Schema, Cadence, Alerts, and Governance

Deploying a monitoring solution in a regulated environment goes beyond just calculations – it requires operationalizing it in a reliable, governed way. Here we outline best practices around **metric schema standardization**, monitoring **cadences**, alerting strategies, and **data governance** concerns, all tailored to financial services contexts.

* **Standardized Metric Schema:** Define a clear schema for storing and reporting metrics. This could mean having separate tables or structured records for *data profile metrics* vs *drift metrics*, with consistent columns such as model\_id, feature\_name, window\_time, metric\_name, value, etc. A standardized schema (like Databricks’ division into profile\_metrics and drift\_metrics tables[[18]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=When%20a%20monitor%20runs%20on,and%20a%20drift%20metrics%20table)) makes it easier to query and automate reports. It also ensures that as models are added or replaced, all follow the same monitoring format (facilitating enterprise-wide dashboards). Regulators value consistency – e.g., an internal model risk report might have a template showing PSI for top 10 features of every high-risk model. By standardizing, you can populate such templates directly from the monitoring system. Additionally, include **metadata** like model version, data source, and possibly data sample size for each metric (so confidence in metrics can be gauged – a drift metric on 100 data points vs 10,000 is different).
* **Monitoring Cadence and Windows:** Choosing how frequently to calculate metrics (the time window or batch size) is crucial. Best practice is to align the cadence with both **data velocity** and **risk tolerance**. For fast-changing data (e.g. transaction fraud), a daily or even hourly window might be needed; for slow-changing ones (e.g. quarterly PD models), monthly might suffice. However, beware of too-small windows: *“when chunks are too small, statistical results become unreliable… what looks like drift may be just noise”*[[64]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=you%20want%20to%20monitor%20or,that%20we%20get%20reliable%20results). A general guideline is each monitoring window should have enough samples to make metrics statistically meaningful (for instance, at least a few hundred observations for a credit model’s monthly PSI). Many firms set a **rolling window** approach – e.g., compute metrics weekly, but also look at trailing 3-month metrics for trend. The cadences often follow business cycles: end-of-month for finance, end-of-day for trading, etc., to integrate with reporting routines. It’s also wise to perform *aggregated quarterly reviews* even if monthly monitoring is in place, to see the bigger picture without daily noise. Document the chosen cadence in policy – e.g., “Data drift metrics will be computed weekly and reviewed by the model owner, with a formal report to risk committee quarterly.” This becomes part of model governance documentation.
* **Thresholds and Alerting Best Practices:** Alerts should be designed to catch meaningful issues without overwhelming stakeholders. Many experts suggest starting with **simple statistical thresholds** on key metrics as an initial approach[[65]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=begin%20to%20determine%20if%20your,start%20monitoring%20your%20ML%20service). For example:
* PSI > 0.2 on any input triggers a warning; PSI > 0.3 triggers a high-severity alert.
* AUC drop below 0.70 triggers alert, or more than 5% drop from baseline performance triggers alert.
* Chi-square test p-value < 0.01 triggers alert for categorical drift.

It’s useful to categorize alerts (e.g., yellow vs red status) to prioritize responses. Use domain knowledge: for instance, if *interest rate* distribution drifts, that might be expected due to market changes (no alert unless performance is hit), whereas drift in *application channel* might be less expected and worth an investigation. As a baseline, alert on what’s likely to break the model. Some banks implement a **“three strikes” rule** – an alert that persists three periods in a row triggers model redevelopment. Others use an **alert matrix**, where multiple metrics in alert together escalate the severity. Modern platforms allow complex conditions (e.g., alert if *both* drift metric and performance metric exceed thresholds, indicating it’s a drift that matters[[61]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Most%20people%20and%20tools%20assume,drift%20can%20be%20more%20important)). This can reduce false positives. Always test and calibrate thresholds on historical data if possible, to see how often they’d have fired. Update thresholds as models evolve or as more data is collected. And ensure alerts are actionable – every alert should have an owner and a playbook (e.g., if concept drift alert: verify if data pipeline changed, consider refreshing model with latest data, etc.).

* **Alert Routing and Incident Management:** In a financial institution, model monitoring alerts should integrate with existing incident management. For example, if using Grafana or another tool, route critical alerts to on-call engineers or a specific Model Risk mailbox. Define SLAs – e.g., *model owner must respond within X hours to any high-sev drift alert*. Incorporate alerts into periodic risk reports as well: for instance, a monthly risk dashboard to senior management could show “Model X triggered 2 alerts this month (data drift in variable Age, concept drift in model accuracy) and actions taken.” This assures management that the monitoring system is not only detecting but also driving responses.
* **Data Governance and Privacy:** Financial data is sensitive. When setting up monitoring, one must ensure compliance with data protection (GDPR, banking secrecy, etc.). **Anonymize or aggregate** monitoring data – drift metrics generally don’t require customer identifiers, so avoid including PII in any logging. If using a third-party SaaS for monitoring (like Arize or WhyLabs cloud), be extremely cautious: many banks prefer self-hosted to keep data in-house. If data must be sent out, consider sending only summary statistics (e.g., distribution bins) rather than raw records. Also, set retention policies: how long are you keeping the monitoring data? Model decisions might be contested legally years later, and monitoring logs could be evidence of due diligence. Many keep at least as long as the model is in production plus some years archive. Ensure that data retention aligns with your overall data governance (some jurisdictions require not keeping data too long, so maybe aggregate or delete detailed logs after a period).
* **Access Control:** Not everyone should see all models’ details. For example, a model used in anti-money laundering might be highly confidential. Use authentication/authorization on dashboards (Grafana supports roles, Databricks can restrict queries by ACLs, etc.). If using MLflow, consider an authenticated backing store or run separate tracking servers per department to silo access. Logs and metrics may indirectly reveal business information (e.g., an increase in default rate could be market-sensitive), so treat monitoring outputs with similar confidentiality as the model’s inputs/outputs.
* **Model Governance Integration:** Embed monitoring into the governance process. This means when a model is approved by a Model Risk Committee, the approval comes with a monitoring plan (which metrics, what thresholds, who monitors, etc.). That plan should be stored in the model inventory. Then, actual monitoring results should periodically be reviewed by governance forums. Some organizations have a monthly “Model Performance Review” meeting with stakeholders from model development, validation, business, and risk – going over key drift and performance metrics for each model. This fulfills oversight requirements. If issues are found, document the remediation (e.g., model will be retrained next week; in the meantime, maybe put a safeguard or increased human review on decisions).
* **Continuous Improvement:** Use the data from monitoring not just for alerts, but to improve models and data pipelines. For example, if a certain data drift keeps occurring, maybe the data integration process needs fixing or the model needs to be retrained more often. Over time, you might also refine which metrics truly correlate with bad outcomes. Perhaps you find PSI on feature X moves a lot but doesn’t affect performance – you might stop alerting on X to focus on more impactful metrics. This feedback loop makes the monitoring more efficient and trusted by stakeholders (they won’t get numb to alerts that don’t matter).
* **Regulatory Compliance and Auditability:** Ensure that the monitoring system itself is auditable. Keep records of metric definitions, threshold rationale, and any changes to them. If you adjust an alert threshold, log why (maybe in a model risk management system). Regulators or internal audit might ask: *“Your model’s accuracy dropped in Q2 last year, what was done?”* – you should be able to show the alert fired, the issue was investigated (perhaps concept drift due to pandemic effects), and the action (model recalibrated or policy adjusted) was taken. Having a tracking ticket or report for that is ideal. Some banks map monitoring to regulatory requirements, for example tagging certain metrics as addressing *“ongoing monitoring”* as per SR 11-7 or EBA guidelines, to demonstrate compliance explicitly.
* **Tooling Choices and Vendor Documentation:** When using vendor tools, leverage their documentation and whitepapers for best practices. For instance, Arize provides templates and dashboards for common financial use-cases (credit, fraud) – using these out-of-the-box can align with industry practice. WhyLabs or Fiddler have “monitoring policy” features to codify thresholds; adopting those ensures consistency. Keep an eye on evolving standards (e.g., upcoming ISO standards on AI performance monitoring) which may define common metrics or processes – aligning early can save future rework.

In conclusion, **operationalizing model monitoring in finance** means treating it as a first-class component of the ML lifecycle, with the same rigor as model development or validation. Standard schemas enable enterprise-wide analysis, proper cadence and threshold setting ensure meaningful signals, and governance integration ensures accountability. By following these practices, an organization not only keeps models accurate and reliable but also builds evidence that it’s fulfilling its fiduciary and regulatory responsibilities in managing model risk. As the ECB noted, *“ongoing monitoring… allows for timely discovery of issues, necessary adjustments of models, and compliance with both internal policies and external regulation”*[[66]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L543%20Ongoing%20monitoring,It%20is%20important)[[67]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=Ongoing%20monitoring%20establishes%20a%20framework,It%20is%20important) – this is ultimately the goal of a robust model monitoring framework in banking.

## Implementation Recommendations for Financial Services Environments

Finally, synthesizing the above, here are **recommendations** for implementing model monitoring and drift management tailored to financial services, where risk and compliance are paramount:

1. **Start Simple, Then Layer Complexity:** Implement baseline monitoring quickly using available tools – e.g., enable Databricks Lakehouse Monitoring if you’re on Databricks, or set up a basic Prometheus/Grafana with a few key metrics. Use simple thresholds (perhaps based on expert judgment or training variance) to get initial alerts. Over time, refine the approach: add more features to monitor, incorporate multivariate methods, adjust thresholds based on observed behavior. This incremental approach gets you coverage early (satisfying regulators that at least something is in place) and avoids analysis paralysis in designing the “perfect” system from day one.
2. **Use Open-Source Libraries for Rapid Deployment:** Tools like **Evidently AI, NannyML, or Deepchecks** can compute a wide range of drift and performance metrics with minimal coding. They can output ready-made reports (which can be logged to MLflow or displayed in dashboards). For example, Evidently’s presets can give you data drift and integrity reports that cover most of what you need[[68]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=report%20%3D%20Report%28metrics%3D)[[69]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=Image%3A%20Evidently%20distribution%20drift%20for,individual%20features). This accelerates implementation and ensures you're using well-tested statistical methods (trusted by the open-source community). You can then integrate their outputs into your architecture of choice (e.g., run Evidently in a Databricks notebook and store results, or run it in a nightly job that pushes metrics to Prometheus). Many banks prototype with open-source libraries and only later transition to a fully integrated platform once requirements stabilize.
3. **Design for Segregation of Environments:** In banking, separate dev, test, and prod environments are the norm. Ensure your monitoring solution respects this – e.g., metrics from model development experiments shouldn’t mix with production metrics. For instance, if using MLflow, have a dedicated tracking server (or at least separate experiment namespace) for production monitoring. If using Grafana, have a separate dashboard folder or instance for production vs pre-production models. This also allows testing the monitoring setup with dummy data or in UAT before affecting live monitoring. It’s crucial that any **test alerts** do not get routed to execs or regulators by mistake, to avoid unnecessary panic.
4. **Implement Monitoring as Code and Pipeline:** Wherever possible, treat the monitoring configuration as code – check it into version control. For example, if you use Grafana, you can define the dashboard JSON and alert rules in code. If using custom Python for metrics, that code should be in a repository with reviews. This ensures changes are tracked and approved. It also helps with **reproducibility** – if an issue occurred, you can recreate what the monitoring logic was at that time. Use CI/CD for the monitoring infrastructure itself (e.g., deploy Prometheus and Grafana via infrastructure-as-code, deploy monitoring jobs via Airflow or similar). This approach increases reliability and auditability.
5. **Mermaid Diagram – Overall Workflow:** The following diagram outlines an example end-to-end workflow combining several best practices discussed:

flowchart TD  
 subgraph DataAndModelPipeline  
 A[Production Data] -->|new batch / stream| B(Model Prediction)  
 B --> C[Prediction Log]  
 B --> D[Feedback Log (with actual outcomes)]  
 end  
 subgraph MonitoringPipeline  
 C --> E[Data Drift Computation]  
 D --> F[Performance Computation]  
 E --> G[Metrics Store / Dashboard]  
 F --> G[Metrics Store / Dashboard]  
 G --> H{{Alerts & Reports}}  
 end  
 subgraph Governance  
 H --> I[Model Owner]  
 H --> J[Risk Manager]  
 I -->|Review issues| K[Retrain/Adjust Model]  
 J -->|Report status| L[Risk Committee]  
 end

*Figure 3: End-to-end monitoring and governance workflow. Production data and model predictions feed into drift and performance computations. Metrics and alerts are surfaced on dashboards. Model owners and risk managers jointly review alerts – model owners handle technical fixes (retraining, etc.) while risk managers ensure compliance (reporting to committees, documentation). This aligns technical monitoring with the bank’s governance structure.*

This diagram highlights how the **technical monitoring loop** (data -> metrics -> alert) ties into the **governance loop** (alert -> human review -> action -> oversight). Ensuring both loops function and connect is key in financial services.

1. **Engage Model Risk Management Early:** Involve the risk governance team in designing the monitoring. They can provide insights on what metrics they expect to see, and they will be the ones consuming the monitoring reports for validation purposes. For instance, agree on which metrics constitute *“key performance indicators”* for each model and need to be included in the official model monitoring report. By engaging them, you also train them on the new tools (if you introduce Grafana dashboards to risk officers, give them a tutorial). Their buy-in will make approvals smoother and oversight more effective.
2. **Plan for Scaling and Multiple Models:** As you implement for one model, consider the future where dozens of models will be monitored. Design your solution to be multi-model from the start: use model identifiers in metrics, template-ize dashboards (Grafana can use template variables to switch model). Also consider aggregation: a *portfolio view* dashboard that shows the status of all models (perhaps a table with green/yellow/red status for each) is very useful for senior management. Scaling also means performance: ensure your metrics store and processing can handle the load (maybe start with weekly offline processing, but architect such that moving to daily or real-time won’t require a complete rework).
3. **Documentation and Training:** Document the monitoring design in a “Model Monitoring Framework” document or Confluence page (perhaps an artifact this very solutioning document can seed). Include: metrics definitions, thresholds, roles and responsibilities, and examples of interpretation. Also, train the users – the data scientists interpreting drift, the validators reviewing reports, and even business users if they need to understand what an alert means. For example, a branch manager might want to know if a credit model in their domain has drifted – you may provide a simple view or report for them, with an explanation that “feature X distribution changed due to policy Y, but model still okay” or similar. Education ensures that when drift is detected, stakeholders react appropriately and not out of misunderstanding.
4. **Test the Monitoring System:** Conduct simulations to verify the monitoring catches what it should. Intentionally introduce drift in a controlled setting (e.g., modify input data in a copy dataset) to see if alerts fire. Also test the end-to-end: if an alert fires, does the right person get notified and do they know what to do? This is akin to a fire drill – extremely useful in a regulated environment to demonstrate the system works. You can document these test results as evidence for auditors that “our monitoring system was tested on scenario X and successfully detected the issue within Y time.”
5. **Stay Updated with Regulatory Changes:** The landscape of AI risk regulation is evolving (EU AI Act, updated Basel guidance for AI/ML, etc.). Continuously update your monitoring to meet any new requirements. For example, if new guidelines ask for *fairness monitoring* or *explainability monitoring*, incorporate those (monitor bias metrics or stability of feature importance over time). Being proactive will keep your institution ahead of compliance mandates and avoid scramble later.

By following these recommendations, a financial institution can implement a **comprehensive, compliant, and responsive** model monitoring capability. This not only guards against model failures and drift-driven losses, but also instills confidence in regulators and stakeholders that the institution is in control of its AI/ML tools. Monitoring is not just a technical necessity; it’s a linchpin of **trustworthy AI in finance**, ensuring models continue to earn the trust placed in them by businesses, customers, and regulators alike.

**Sources:**  
- Basel Committee on Banking Supervision – *Sound Practices for Model Risk Management* (Ongoing Monitoring)[[1]](https://empoweredsystems.com/blog/what-is-the-basel-morm-framework-for-model-risk-management/#:~:text=all%20technical%20requirements,analysis%20and%20corrective%20action%20plans)  
- Bank of England Prudential Regulation – *Principles for AI/ML Model Risk* (Monitoring expectations)[[5]](https://www.bankofengland.co.uk/prudential-regulation/publication/2023/may/model-risk-management-principles-for-banks#:~:text=,%E2%80%93%20such%20ethical%20challenges%20could)  
- ISO/IEC 42001:2023 – *AI Management System* (Continuous monitoring requirements)[[6]](https://www.brightdefense.com/resources/iso-42001-compliance/#:~:text=Organizations%20must%20establish%20data%20collection%2C,to%20operate%20safely%20and%20ethically)  
- NIST AI Risk Management Framework 1.0 (AI RMF) – *Manage function (continuous monitoring)*[[7]](https://www.diligent.com/resources/blog/nist-ai-risk-management-framework#:~:text=monitoring%20processes%20to%20reduce%20identified,roles%20for%20ongoing%20AI%20risk)  
- Databricks Documentation – *Lakehouse Monitoring* (data & drift metrics, PSI thresholds)[[21]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=js_distance%20double%20Jensen%E2%80%93Shannon%20distance%20for,for%20categorical%20columns)[[22]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=,2%20indicates%20significant%20population%20change)  
- Grafana Labs Blog – *BasisAI case study* (Prometheus+Grafana for model drift, Boxkite, KL divergence & KS test)[[33]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=other%20hand%2C%20shows%20no%20visible,difference)[[34]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Conclusion)  
- NannyML Blog – *Data Drift vs. Performance* (why monitor performance, chunk size effects)[[50]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=1,shift%20occurs%20in%20similar%20regions)[[64]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=you%20want%20to%20monitor%20or,that%20we%20get%20reliable%20results)  
- Evidently AI Tutorial – *ML Monitoring Guide* (logging reports to MLflow, trend tracking)[[36]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=You%20can%20go%20pretty%20far,is%20detected%20in%20your%20dataset)[[37]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=Image%3A%20Evidently%20Reports)  
- Arize AI Guides – *Model Drift Metrics* (PSI, JS, EMD definitions)[[42]](https://arize.com/model-drift/#:~:text=monitor%20and%20troubleshoot%20drift%2C%20including,EMD)[[40]](https://arize.com/model-drift/#:~:text=1,Expected)  
- BentoML Blog – *Guide to Drift Detection* (threshold alert recommendation, concept drift example)[[65]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=begin%20to%20determine%20if%20your,start%20monitoring%20your%20ML%20service)[[35]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=correct%20predictions%20that%20the%20model,is%20generating%20begins%20to%20degrade)  
- Grant Thornton – *Ongoing Monitoring in MRM* (SR 11-7 compliance, data quality in monitoring)[[3]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L584%20In%20accordance,It)[[70]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=In%20accordance%20with%20SR%2011,It)

[[1]](https://empoweredsystems.com/blog/what-is-the-basel-morm-framework-for-model-risk-management/#:~:text=all%20technical%20requirements,analysis%20and%20corrective%20action%20plans) What is the Basel MoRM Framework for Model Risk Management? | Empowered GRC Platform | Streamline Governance, Risk & Compliance

<https://empoweredsystems.com/blog/what-is-the-basel-morm-framework-for-model-risk-management/>

[[2]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L543%20Ongoing%20monitoring,It%20is%20important) [[3]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L584%20In%20accordance,It) [[4]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=Weakness%20in%20an%20organization%E2%80%99s%20ongoing,Risk) [[8]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L665%20actively%20managed,escalated%20up%20to%20the%20board) [[14]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=Ongoing%20monitoring%20establishes%20a%20framework,It%20is%20important) [[66]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=match%20at%20L543%20Ongoing%20monitoring,It%20is%20important) [[67]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=Ongoing%20monitoring%20establishes%20a%20framework,It%20is%20important) [[70]](https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management#:~:text=In%20accordance%20with%20SR%2011,It) Facilitating ongoing monitoring in Model Risk Management | Grant Thornton

<https://www.grantthornton.com/insights/articles/banking/2023/facilitating-ongoing-monitoring-in-model-risk-management>

[[5]](https://www.bankofengland.co.uk/prudential-regulation/publication/2023/may/model-risk-management-principles-for-banks#:~:text=,%E2%80%93%20such%20ethical%20challenges%20could) [[11]](https://www.bankofengland.co.uk/prudential-regulation/publication/2023/may/model-risk-management-principles-for-banks#:~:text=,the%20lowest%20tier%20models%3B%20and) PS6/23 – Model risk management principles for banks | Bank of England

<https://www.bankofengland.co.uk/prudential-regulation/publication/2023/may/model-risk-management-principles-for-banks>

[[6]](https://www.brightdefense.com/resources/iso-42001-compliance/#:~:text=Organizations%20must%20establish%20data%20collection%2C,to%20operate%20safely%20and%20ethically) [[13]](https://www.brightdefense.com/resources/iso-42001-compliance/#:~:text=%2A%20Automatic%20Decision,ISO%2042001%20mandates%20rigorous%20data) ISO 42001: A Framework for Ethical and Responsible AI Systems

<https://www.brightdefense.com/resources/iso-42001-compliance/>

[[7]](https://www.diligent.com/resources/blog/nist-ai-risk-management-framework#:~:text=monitoring%20processes%20to%20reduce%20identified,roles%20for%20ongoing%20AI%20risk) [[12]](https://www.diligent.com/resources/blog/nist-ai-risk-management-framework#:~:text=match%20at%20L427%20or%20transaction,to%20detect%20discriminatory%20outcomes%2C%20monitor) NIST AI Risk Management Framework: A simple guide to smarter AI governance

<https://www.diligent.com/resources/blog/nist-ai-risk-management-framework>

[[9]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Many%20people%20and%20tools%20advocate,with%20the%20model%E2%80%99s%20test%20set) [[10]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=I%20ran%20some%20experiments%20using,as%20the%20reference%20for%20monitoring) [[48]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=NannyML%20has%20two%20methods%20to,in%20this%20other%20%2022) [[49]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Red%20flag%204%3A%20focusing%20only,and%20not%20on%20model%20performance) [[50]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=1,shift%20occurs%20in%20similar%20regions) [[57]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=I%20used%20one%20of%20NannyML%E2%80%99s,my%20surprise%2C%20there%20was%20none) [[58]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Under%20the%20hood%2C%20DLE%20trains,provides%20us%20with%20a%20way) [[59]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=none) [[60]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=This%21%20This%20is%20a%20really,Performance) [[61]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=Most%20people%20and%20tools%20assume,drift%20can%20be%20more%20important) [[64]](https://www.nannyml.com/blog/data-drift-does-it-matter#:~:text=you%20want%20to%20monitor%20or,that%20we%20get%20reliable%20results) There's Data Drift, But Does It Matter?

<https://www.nannyml.com/blog/data-drift-does-it-matter>

[[15]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=Databricks%20Lakehouse%20Monitoring%20lets%20you,data%20quality%20and%20model%20performance) [[16]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=,performing%20better%20than%20version%20B) [[17]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=To%20draw%20useful%20insights%20from,help%20you%20identify%20the%20cause) [[27]](https://docs.databricks.com/aws/en/lakehouse-monitoring/#:~:text=,contact%20your%20Databricks%20account%20team) Introduction to Databricks Lakehouse Monitoring | Databricks on AWS

<https://docs.databricks.com/aws/en/lakehouse-monitoring/>

[[18]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=When%20a%20monitor%20runs%20on,and%20a%20drift%20metrics%20table) [[19]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=the%20metrics%20in%20the%20row,null%20values) [[20]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=chi_squared_test%20%60struct,distance%20for%20drift%20in%20distribution) [[21]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=js_distance%20double%20Jensen%E2%80%93Shannon%20distance%20for,for%20categorical%20columns) [[22]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=,2%20indicates%20significant%20population%20change) [[23]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=Grouping%20columns%20window%20%60struct,Value%20of%20the%20slicing%20expression) [[24]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=the%20metrics%20in%20the%20row,Number%20of%20columns%20with) [[25]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=%60slicing_exprs%3D%5B) [[26]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=Additional%20statistics%20are%20calculated%20for,analysis) [[43]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=non_null_columns_delta%20%60struct,for%20categorical%20columns) [[51]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=comparison%20window,square%20test%20for) [[53]](https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output#:~:text=Column%20name%20Type%20Description%20Grouping,7) Monitor metric tables - Azure Databricks | Microsoft Learn

<https://learn.microsoft.com/en-us/azure/databricks/lakehouse-monitoring/monitor-output>

[[28]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Boxkite%2C%20a%20simple%2C%20open%20source,distribution%20between%20the%20two%20environments) [[29]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Under%20the%20hood%2C%20Boxkite%20leverages,for%20Prometheus%20to%20periodically%20scrape) [[30]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=match%20at%20L547%20Under%20the,for%20Prometheus%20to%20periodically%20scrape) [[31]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=After%20setting%20up%20Prometheus%20as,propagated%20to%20the%20query%20string) [[32]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Grafana%E2%80%99s%20histogram%20visualization%20helps%20us,hand%2C%20shows%20no%20visible%20difference) [[33]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=other%20hand%2C%20shows%20no%20visible,difference) [[34]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=Conclusion) [[55]](https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/#:~:text=We%20use%20two%20metrics%20to,explained%20in%20our%20demo%20video) How BasisAI uses Grafana and Prometheus to monitor model drift in machine learning workloads | Grafana Labs

<https://grafana.com/blog/2021/08/02/how-basisai-uses-grafana-and-prometheus-to-monitor-model-drift-in-machine-learning-workloads/>

[[35]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=correct%20predictions%20that%20the%20model,is%20generating%20begins%20to%20degrade) [[56]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=Simple%20strategies%20such%20as%20monitoring,cost%20way%20to%20start%20monitoring) [[62]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=thresholds%20is%20a%20great%20way,Simple) [[65]](https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection#:~:text=begin%20to%20determine%20if%20your,start%20monitoring%20your%20ML%20service) A Guide To ML Monitoring And Drift Detection

<https://www.bentoml.com/blog/a-guide-to-ml-monitoring-and-drift-detection>

[[36]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=You%20can%20go%20pretty%20far,is%20detected%20in%20your%20dataset) [[37]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=Image%3A%20Evidently%20Reports) [[52]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=Once%20you%20run%20the%20code%2C,it%20up%20in%20a%20report) [[68]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=report%20%3D%20Report%28metrics%3D) [[69]](https://www.evidentlyai.com/blog/mlops-monitoring#:~:text=Image%3A%20Evidently%20distribution%20drift%20for,individual%20features) How to start with ML model monitoring. A step-by-step guide.

<https://www.evidentlyai.com/blog/mlops-monitoring>

[[38]](https://blog.devgenius.io/mlflow-for-model-monitoring-cb8b2177b67a?gi=8792005f53bb#:~:text=Conclusion) MLflow for model monitoring. Introduction | by Vitomir Jovanović | Dev Genius

<https://blog.devgenius.io/mlflow-for-model-monitoring-cb8b2177b67a?gi=8792005f53bb>

[[39]](https://nannyml.readthedocs.io/en/main/tutorials/detecting_data_drift.html#:~:text=The%20model%20has%20been%20trained,X) [[54]](https://nannyml.readthedocs.io/en/main/tutorials/detecting_data_drift.html#:~:text=Data%20drift%20is%20one%20of,drift%20and%20concept%20drift%20simultaneously) Detecting Data Drift — NannyML 0.13.1 documentation

<https://nannyml.readthedocs.io/en/main/tutorials/detecting_data_drift.html>

[[40]](https://arize.com/model-drift/#:~:text=1,Expected) [[41]](https://arize.com/model-drift/#:~:text=these%20cause%20issues%20with%20KL,euclidean%20distance%20check%20determines%20if) [[42]](https://arize.com/model-drift/#:~:text=monitor%20and%20troubleshoot%20drift%2C%20including,EMD) Model Drift & Machine Learning: Concept Drift, Feature Drift, Etc.

<https://arize.com/model-drift/>

[[44]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=Yet%2C%20due%20to%20the%20multidimensional,Guide%20to%20the%20Multivariate%20Approach) [[45]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=Multivariate) [[46]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=Data%20Reconstruction%20with%20PCA) [[47]](https://www.nannyml.com/blog/monitoring-data-drift#:~:text=The%20differences%20between%20these%20two,detection%20mechanisms%20are%20summarized%20below) A detailed overview of univariate and multivariate data drift methods for OSS and cloud implementation.

<https://www.nannyml.com/blog/monitoring-data-drift>

[[63]](https://snorkel.ai/blog/how-to-apply-and-use-machine-learning-observability/#:~:text=Arize%20AI%20on%20How%20to,data%20drift%2C%20and%20concept%20drift) Arize AI on How to apply and use machine learning observability

<https://snorkel.ai/blog/how-to-apply-and-use-machine-learning-observability/>