Balancing Efficiency and Sensitivity in Embedding-Based Concept Drift Detection for Deep Learning

Jasper Bruin

Supervisors:

Jan Rellermeyer

J.S.Rellermeyer@tudelft.nl

Asterios Katsifodimos

A.Katsifodimos@tudelft.nl

TU Delft

May 2025

Abstract

This thesis investigates the effectiveness and efficiency of embedding-based drift detection in machine learning systems, focusing on synthetic simulations and real-world production data. Through controlled experiments, we compare vector-based and distribution-based metrics regarding sensitivity to drift, memory and runtime cost, and practical utility for early warning of performance degradation. Results from synthetic drift experiments indicate that vector-based metrics respond quickly to small shifts but tend to saturate early, limiting their ability to differentiate between moderate and severe drift. Distribution-based metrics, by contrast, scale more proportionally across the entire drift spectrum, providing more stable and interpretable signals. Memory and runtime profiling show that vector-based methods are consistently more efficient, while distribution-based approaches incur higher costs. A real-world evaluation using eight years of data from a deployed recommendation system confirms the practical value of these findings. Vector metrics consistently provided earlier signals, on average 87 days before performance drops, compared to distribution metrics, which often lagged. However, distribution metrics offered smoother trends and fewer false positives, making them better suited for long-term monitoring. This thesis also explores trade-offs introduced by embedding compression. Principal Component Analysis (PCA) and KLL Sketches were evaluated for reducing computational overhead. PCA preserves the drift signal better in vector metrics, but is more resource-intensive. In contrast, KLL is highly efficient, but sacrifices sensitivity, particularly in vector space.

Contents

Chapter 1

Introduction

Machine learning systems are deployed in dynamic environments where the statistical properties of input data change over time. These shifts, caused by evolving user behaviour, seasonal patterns, or external events, violate the assumption of stationarity and can gradually degrade model performance. To detect such changes, modern systems monitor the internal embeddings of deep networks and raise alerts when distances or divergences from reference distributions increase. These embedding-level signals are computationally efficient and empirically sensitive. However, it remains unclear whether they consistently indicate harmful concept drift or merely reflect harmless representation shifts.

This study focuses on two main types of embedding-based drift detection methods. **Vector-based** methods compute distances between individual embedding vectors, compact numerical representations of inputs, using metrics such as Euclidean or Mahalanobis distance. These methods are sensitive to small, instance-level changes. In contrast, **distribution-based** methods compare complete distributions of embeddings over time using statistical measures such as the Wasserstein distance or Maximum Mean Discrepancy (MMD), capturing broader trends in feature space.

Despite increasing interest in embedding-based and distance-aware detection techniques due to the growing use in Large Language Models [5, 7, 34], a question remains: How useful are these detected representation shifts in practice? While many modern approaches raise drift alarms based on high-dimensional deviations, recent evidence suggests that even significant shifts in feature space may have little effect on model performance [20]. While Chen et al. highlights feature-space drift as less impactful, it does not specifically dive into vector-based distance metrics in embedding spaces. This is important for understanding the implications of representation shifts for meaningful concept drift detection. There's a gap between benign and harmful shifts in the embedding space across dynamic data streams. The research only investigated how MVP models behave in the feature space, such as LinearSVM and SecSVM. Understanding how model size impacts computational overhead and exploring architectural optimisations remains unexplored within the scope of these papers [5, 7, 34]. Evaluating the trade-offs between computational efficiency and similarity preservation through embedding reduction techniques is an area not covered by papers that are currently published. Current research also lacks insight into how embedding extraction across various architectures confirms stability across feature-based metrics.

Although these advancements exist, a considerable research gap remains: The systematic understanding of how vector-based distance metrics relate to concept drift in dynamic data streams is still lacking. Current literature does not sufficiently address the impact of model architectural optimisations on mitigating computational overhead, nor does it thoroughly explore the trade-offs between embedding reduction techniques such as PCA and KLL Sketch regarding computational efficiency and similarity preservation [74, 40]. Moreover, while various distance metrics (e.g., Mahalanobis, Euclidean, Canberra) are employed for measuring similarity drift, there is inadequate investigation into their reliability when coupled with different embedding techniques across diverse application scenarios. This presents a need for robust benchmarks and comparative studies focusing on the efficiency and effectiveness of drift detection, particularly highlighting the balance between computational efficiency and similarity retention across different embedding techniques and distance metrics.

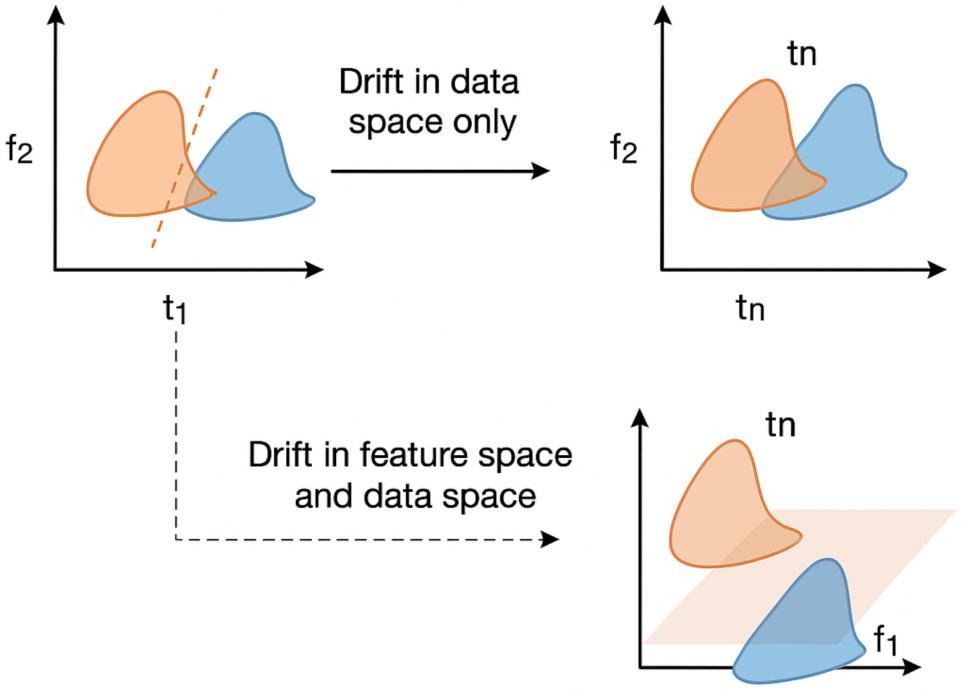


Figure 1.1: The Difference between drift and data space is illustrated in embedded space.

Much of the current evaluation of drift detectors is limited to synthetic or low-dimensional data. However, real-world data streams, particularly those generated by large language models (LLMs) and deep learning (DL) architectures, present additional challenges such as semantic variability, continuous embedding drift, and computational cost. A recent benchmarking effort identifies several limitations in existing detectors when applied to these settings and calls for more empirical research under operational constraints [47].

Reliable concept drift detection also holds significant societal and practical value outside

these technical issues. Detecting distribution changes early in healthcare can prevent diagnostic systems from making incorrect or outdated predictions [62]. In financial systems, drift-aware models are better equipped to detect fraud patterns in real-time [15]. In e-commerce, timely drift detection helps recommender systems adapt to user behaviour, reducing performance degradation [49, 37]. As machine learning increasingly supports high-impact decisions, efficient and robust detection methods are essential to ensure safety, fairness, and system integrity.

This thesis addresses these challenges by empirically evaluating how well vector-based metrics (e.g., Euclidean, Mahalanobis) and distribution-based divergences (e.g., Wasserstein, MMD) identify performance-relevant concept drift in embedding spaces. It also investigates distinguishing harmful drift from benign shifts in streaming environments. Additionally, we compare two embedding reduction strategies, Principal Component Analysis (PCA), a linear compression technique, and KLL Sketch, a probabilistic summarisation method, to understand the trade-offs between memory efficiency and semantic retention [74, 59]. The goal is to provide a practical and interpretable framework for detecting drift in complex, high-dimensional systems.

Research Objectives and Questions. The main objectives of this thesis are to assess the effectiveness of embedding-based drift detection in identifying performance-affecting drift and to distinguish between harmful and benign changes in dynamic input streams. This includes evaluating the trade-offs between detection sensitivity and computational cost using embedding compression techniques. It also considers how different model architectures and embedding extraction methods affect detection robustness. The following questions guide this research:

1. To what extent do shifts detected using vector-based distance metrics in embedding spaces correspond to meaningful performance-impacting concept drift, and how can we distinguish between benign and harmful representation changes in dynamic data streams?

2. How effective and efficient are vector-based distance metrics for drift detection when applied to real-world data streams, and what benchmarks are needed to evaluate their robustness across diverse application scenarios?

3. How does model size impact computational overhead, and what role do architectural optimisations play in mitigating this overhead?

4. What are the trade-offs between embedding reduction techniques (PCA, KLL Sketch) regarding computational efficiency and similarity preservation?

5. How do embedding extraction strategies vary across different model architectures (e.g., autoregressive LLMs, encoder-decoders, masked LLMs, and hybrid models like DeepFM), and what are the implications for similarity retention in drift detection tasks?

6. What is the impact of various distance metrics (e.g., Mahalanobis, Euclidean, Canberra) on measuring similarity drift, and how do different embedding techniques affect their reliability?

7. How do different embedding techniques impact similarity preservation under increasing drift strength across various distance metrics?

8. What are the trade-offs between computational efficiency and similarity retention when using different embedding techniques and distance metrics?

Literature Positioning. Recent work in concept drift detection has shifted toward more practical deployments, especially in unsupervised, high-dimensional environments where labels are unavailable. Werner et al. [70] argue that many existing detectors prioritise accuracy over scalability and neglect performance engineering practices such as runtime profiling and benchmarking. Fuccellaro et al. [27] address this by proposing unsupervised techniques that reduce false positives and separate benign shifts from harmful drift without relying on labelled feedback. In parallel, Chen et al. [19] introduce semantic embedding strategies using knowledge graphs to enhance interpretability and resilience in structured domains.

Large-scale benchmarks, such as those by Lukats et al. [47], show that drift detection methods vary significantly in reliability across real-world datasets, highlighting the importance of consistent evaluation protocols. Wan et al. [69] introduce contrastive embedding methods that improve performance in noisy, high-dimensional settings without labels. Work by Hinder et al. [36] stresses the growing importance of explainability, while Cao et al. [13] show that the choice of embedding technique strongly influences anomaly detection results. Together, these studies motivate the goal of this thesis: to create an efficient and interpretable embedding-based drift detection framework.

Contributions. This thesis presents the *EmbeddingTracker* framework, which combines vector-based and distribution-based drift detection with streaming thresholding capabilities for real-time use. The framework is evaluated using synthetic benchmarks, including controlled token shuffling and feature perturbations, and a real-world stream of Amazon reviews spanning eight years. Key contributions include the finding that, despite a higher false-positive rate, vector-based metrics consistently flag performance, impacting drift earlier than distribution-based alternatives, up to three months in the Amazon case. Additionally, the study shows that KLL Sketch can reduce memory usage by an order of magnitude while preserving 75–90% of full-vector sensitivity. These results provide actionable guidance on designing drift detection pipelines that balance responsiveness and efficiency in real-world systems.

Thesis Structure. The remainder of this thesis is organised as follows, with each chapter building on the previous to develop, evaluate, and reflect on the proposed drift detection framework:

- **Chapter 2** surveys foundational and contemporary work on concept drift and embedding techniques
- **Chapter 3** lays out the mathematical principles underpinning vector- and distribution-based drift detection methods and dimensionality reduction.
- **Chapter 4** introduces the *EmbeddingTracker* framework and details its design and implementation.
- **Chapter 5** describes the experimental setup and benchmark datasets used for evaluation.
- **Chapter 6** presents the results of synthetic and real-world drift detection experiments, including metric sensitivity and memory efficiency analyses.
- **Chapter 7** discusses the implications of the findings, outlines limitations, and explores future directions.
- **Chapter 8** concludes the thesis with a summary of key contributions and takeaways.

Chapter 2

Background & Related Work

2.1 Concept Drift: Definitions and Impact

Concept drift occurs when the statistical properties of data change over time, disrupting the assumption of stationarity in supervised learning systems. Formally, concept drift is defined as a change in the joint probability distribution $P(X, Y)$ over the input variables X and the target variable Y at different time points t_0 and t_1 [29]:

$$P_{t_0}(X, Y) \neq P_{t_1}(X, Y) \quad .$$

This means that the underlying relationship between inputs and outputs shifts, which can degrade model performance. We need strategies that detect these shifts and adapt models accordingly to maintain accuracy.

Concept drift can be quantified by measuring how much the distributions differ. For instance, one can use the Kullback–Leibler (KL) divergence to compare the joint distributions at two time points [43]:

$$D_{KL}(P_{t_0} \parallel P_{t_1}) = \sum_{x,y} P_{t_0}(x, y) \log \frac{P_{t_0}(x, y)}{P_{t_1}(x, y)},$$

for discrete variables or the integral equivalent for continuous variables. A high KL divergence indicates a significant change in distribution. Other metrics like the Wasserstein (Earth Mover's) distance can also measure how much "work" is needed to transform one distribution into another.

These differences in distributions can be framed as hypothesis tests. For example, we can test:

$$H_0 : P_{t_0}(X, Y) = P_{t_1}(X, Y) \quad \text{vs.} \quad H_1 : P_{t_0}(X, Y) \neq P_{t_1}(X, Y).$$

Rejecting H_0 suggests that concept drift has occurred. If we are specifically interested in changes to the conditional distribution, we test:

$$H_0 : P_{t_0}(Y|X) = P_{t_1}(Y|X) \quad \text{vs.} \quad H_1 : P_{t_0}(Y|X) \neq P_{t_1}(Y|X).$$

When the relationship between input features and the target changes, we have what

is known as *real concept drift*. Here, the conditional distribution changes while the input distribution remains the same:

$$P_{t_0}(Y|X) \neq P_{t_1}(Y|X), \quad \text{but} \quad P_{t_0}(X) = P_{t_1}(X).$$

This directly alters decision boundaries and requires immediate adaptation [29]. For example, regulatory changes that modify the relationship between income and default risk would constitute real drift in a credit scoring model.

In contrast, *virtual concept drift* occurs when the marginal input distribution $P(X)$ changes, but the relationship $P(Y|X)$ remains constant:

$$P_{t_0}(X) \neq P_{t_1}(X), \quad \text{but} \quad P_{t_0}(Y|X) = P_{t_1}(Y|X).$$

Unlike real drift, virtual drift does not affect decision boundaries but requires adjustments to handle the changing input distribution. For instance, if a new customer demographic introduces changes in the input data distribution for a credit scoring model but their default behaviour remains consistent, this would represent virtual drift [29].

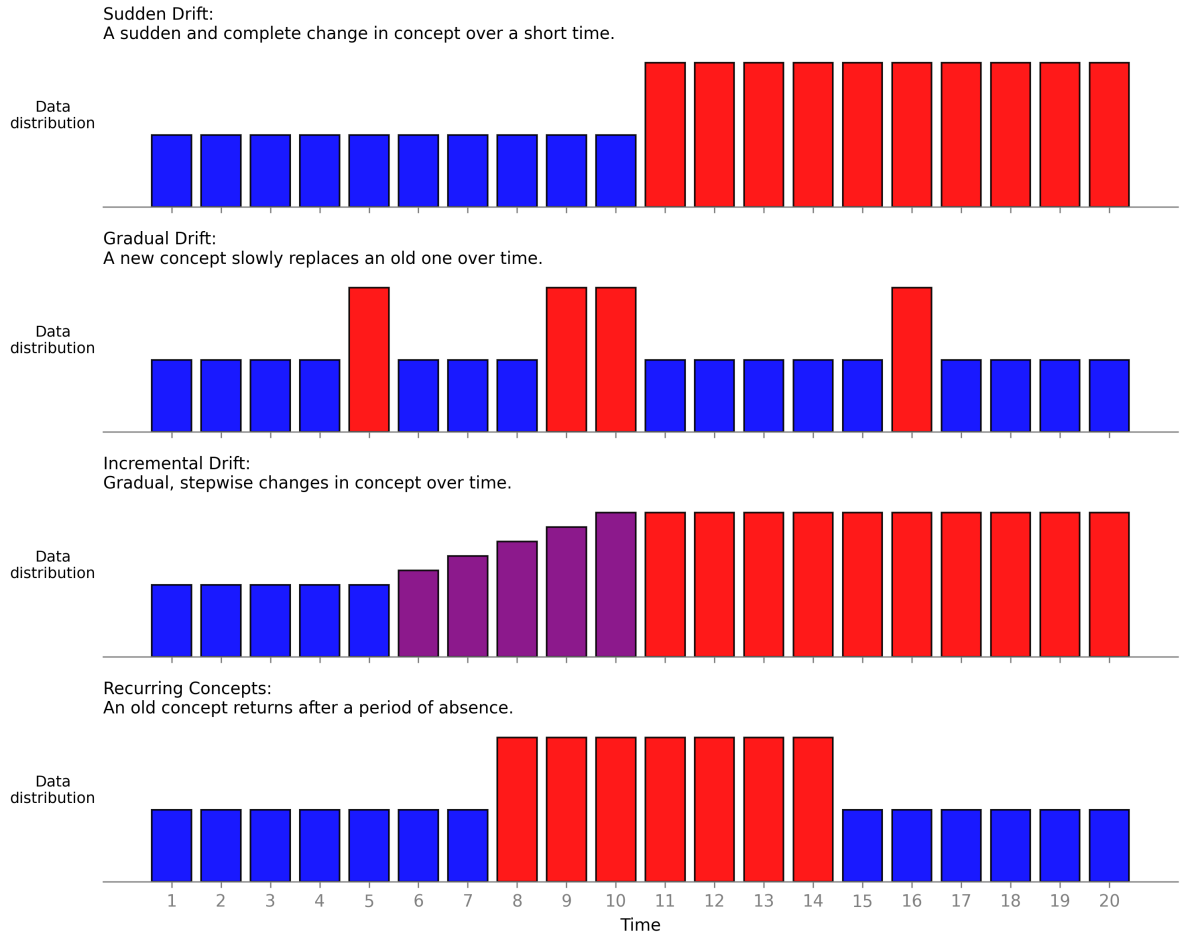


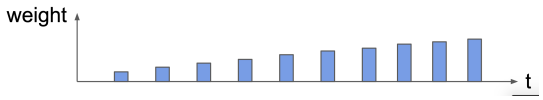
Figure 2.1: Different drift types: (a) Sudden Drift, (b) Gradual Drift, (c) Incremental Drift, and (d) Recurring Concepts.

Concept drift manifests in various forms, as shown in Figure 2.1. *Sudden drift* replaces one concept abruptly [29], *gradual drift* involves a slower shift where old and new concepts overlap [46], and *incremental drift* introduces multiple intermediate states [29]. *Recurring concepts* appear when previous patterns return after a period of absence [46]. These forms can coexist and complicate detection and adaptation.

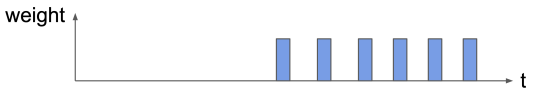
2.1.1 Strategies for Detecting and Adapting to Concept Drift

Adaptive Systems for Managing Drift. Detecting and adapting to drift often involves statistical tests and metrics periodically monitoring differences between distributions or model performance [46]. Simple methods include running hypothesis tests (e.g., Kolmogorov–Smirnov tests) on incoming batches of data to detect significant distributional changes. More advanced methods employ divergence measures like KL divergence or Wasserstein distance to flag shifts. Once drift is detected, adaptive systems classify it (e.g., abrupt vs. gradual) and respond accordingly.

Techniques like sliding windows (Figure 2.2b) focus on retaining recent data in a fixed-



(a) Rewarding more recent data with higher weights



(b) Sliding window focusing on fixed recent data

Figure 2.2: Comparison between weighting recent data more (a) and sliding window methods (b) for adapting to drift.

length window, ensuring the model quickly updates while discarding outdated information [29]. In contrast, approaches that reward more recent data with progressively higher weights (Figure 2.2a) allow for more nuanced adaptation by gradually reducing the importance of older data. Both techniques are critical for handling concept drift and balancing responsiveness and stability. Too much adaptation risks overfitting to temporary changes (excessive plasticity), while too little prevents necessary updates (excessive stability) [29]. By employing quantitative metrics, hypothesis testing, and principled adaptation strategies, systems can address concept drift rigorously, ensuring accurate and reliable performance in dynamic environments [46, 29].

Integrating Drift Detection with Performance Monitoring. Model performance monitoring alone isn't enough to detect data drift effectively [8, 2, 63]. Metrics like AUROC can appear stable even when data distributions shift significantly. Data-driven drift detection methods are better at identifying these shifts. For example, Kore et al. demonstrates that drift can occur without noticeable changes in performance metrics, especially when these metrics fail to capture subtle shifts. This highlights the importance of combining performance monitoring with targeted drift detection techniques. Advanced methods like image-and-output-based drift detection (e.g., TAE + BBSD) have effectively detected natural and synthetic drifts [42]. During the COVID-19 pandemic, these methods identified critical data shifts that traditional metrics overlooked [42]. By detecting distributional changes early, these techniques enable timely re-evaluation and adaptation of models. Combining statistical drift detection with advanced methods ensures models remain accurate and reliable, even in dynamic environments. While not all drifts negatively impact performance, monitoring and adapting models as data evolves is important for maintaining their effectiveness.

2.2 Traditional vs. Modern Drift Detection Approaches

Detecting concept drift is critical for maintaining the robustness and accuracy of machine learning models in non-stationary environments [24, 16, 58, 45, 71]. Choosing the right drift detection method depends on specific challenges, such as label dependencies, computational resource availability, and the type of drift (abrupt, gradual, or recurring). There is no one-size-fits-all solution due to varying domain requirements and constraints, and each method offers unique strengths and limitations. Table 2.1 summarises the key drift detection methods,

highlighting their strengths, weaknesses, and best use cases to assist in selecting an appropriate approach.

- Supervised methods require consistent access to labelled data, making them unsuitable for sparse or delayed feedback environments. For example, these methods excel in real-time systems with reliable label availability.
- Methods like self-supervised learning (SSL) and unsupervised approach strategies can be computationally demanding, requiring careful consideration for real-time or resource-constrained settings, as outlined in Table 2.1.
- The nature of the drift (e.g., abrupt, gradual, or recurring) dictates the optimal method. Ensemble methods are robust to diverse drift types, while the Multi-Armed Bandit (MAB) model approaches efficiently handle abrupt and gradual shifts with resource optimisation.
- Balancing quick adaptation to changes (responsiveness) with maintaining model stability over time is critical. Methods like SSL and semi-supervised approaches offer adaptability but require tuning to prevent overfitting or delayed responses.

Selecting a drift detection method requires careful consideration of the application's constraints and objectives to ensure robust and efficient model performance. Table 2.1 provides a comparative view of the strengths and weaknesses of various approaches.

Method Type	*Strengths*	*Weaknesses*	*Best Use Cases*
Traditional Methods			
Supervised Methods	Directly monitor model performance; effective with immediate labels	Dependence on continuous labels; prone to false positives in noisy environments	Real-time systems with consistent label availability
Unsupervised Methods	Label-independent; early detection of input distribution drift	Computationally intensive; may fail to detect changes in $P(Y\|X)$	Applications with limited or no access to labelled data
Non-Traditional Methods			
Semi-Supervised Methods	Combine labelled and unlabeled data; detect both input and output drift efficiently	Moderate performance tuning required; reliant on pseudo-label quality	Scenarios with scarce labels but high adaptability requirements
Self-Supervised Learning (SSL) Approaches	Exploit inherent data structures; minimal label dependency; suitable for anomaly detection	Drift-specific metrics lacking; computationally expensive for real-time detection	Emerging use cases in domains like computer vision and financial anomaly detection
Ensemble Methods	Robust to diverse drift types; aggregate multiple model predictions	High computational resource usage; complex to implement	Large-scale, resource-rich environments with diverse drift types
Reinforcement Learning (Multi-Armed Bandit Approaches)	Optimized for resource-efficient drift detection; adaptable to evolving environments	Requires careful parameter tuning (e.g., sliding window size); sensitive to configuration	Real-time scenarios with constrained computational resources; suitable for abrupt and gradual drifts

Table 2.1: Summary of Drift Detection Methods

Semi-Supervised Drift Detection Methods

One semi-supervised drift detection method is the Ensemble Semi-Supervised Classification and Regression (ESCR) algorithm, thoroughly discussed by Zheng et al. and inspired by Haque, Khan, and Baron. ESCR begins with a warm-up phase using labelled data to train base classifiers. Each classifier employs clustering techniques to summarise data with pseudo points, including centroids, radii, and class frequencies. These pseudo points represent decision boundaries without storing raw data, ensuring memory efficiency.

As new data instances arrive, ESCR classifies them using majority voting across the ensemble. Confidence scores for each example are calculated and stored in sliding windows. Drift is detected through statistical analysis using Jensen–Shannon divergence, comparing confidence score distributions in consecutive sliding windows [45]. This approach identifies significant changes in data distribution without relying solely on classification error rates. When drift is detected, the algorithm determines if it is recurring by comparing pre- and post-drift distributions. ESCR applies the q-Neighborhood Silhouette Coefficient for novel class detection, measuring cluster cohesion and separation. Instances forming dense clusters far from known

class boundaries are labelled novel classes. To optimise performance, ESCR employs recursive computations and selective execution, reducing the computational burden and allowing the framework to handle high-speed data streams effectively. The ensemble is dynamically updated by replacing older classifiers with new ones trained on labelled and pseudo-labelled data from the current window.

Concerning precision, ESCR outperforms baselines such as ECSMiner and SAND [50, 35]. For example, on the Forest Cover dataset, ESCR achieves an accuracy of 89.7% compared to 86.3% for ECSMiner and 84.9% for SAND. These improvements are consistent across multiple datasets, including synthetic streams like HyperPlane and SynRBF and real-world datasets like PAMAP and Power Supply. Statistical tests like the Friedman test with Nemenyi post hoc analysis confirm ESCR's performance, highlighting its adaptability to various data stream scenarios with different drift dynamics.

Comparing Supervised, Unsupervised, and Semi-Supervised Drift Detection

Supervised methods, such as the *Drift Detection Method (DDM)* [30] and the *Page-Hinckley (PH) Test* [55], are highly effective when immediate access to labelled data is available. However, their performance relies heavily on the continuous availability of labelled data, which may not always be feasible. These methods can also face challenges with gradual drift, as they may require significant memory to manage extended warning phases.

In practical scenarios, variations in data availability and feedback introduce further complexity. For example, the true target values (feedback) may be delayed or unavailable, leading to postponed model updates. In addition, new data points for prediction might arrive before feedback for previously processed data is received, creating a lag in the learning process [68]. Despite these delays, the underlying principles of supervised methods remain consistent. Furthermore, specific environments require data processing in batches rather than incrementally, introducing additional constraints on supervised methods [30]. These challenges highlight the importance of selecting the appropriate method based on the specific characteristics and requirements of the application.

Unsupervised methods such as *Cumulative Sum (CUSUM)* [9] and *Adaptive Windowing (ADWIN)* [8] do not require labelled data and can detect changes in the statistical properties of the input data. However, they may miss changes in the conditional distribution $P(Y|X)$ and can be computationally intensive. Metrics for unsupervised methods, including *Hellinger-Distance*, *Energy-Distance*, and *Wasserstein-Distance*, involve several key considerations. One important factor is computational overhead, which refers to the additional time and resources required for detecting drift. Another critical measure is the false-positive rate (FPR), which is particularly significant for real-time systems to prevent unnecessary alerts. Lastly, drift sensitivity is essential, as it denotes the ability to detect subtle changes in data distribution.

Semi-supervised methods like ESCR balance these approaches, leveraging the strengths

of both supervised and unsupervised techniques. They can detect input and output drift by utilising labelled and unlabelled data while minimising dependency on labelled instances. The choice of method should be guided by the application's data availability, feedback frequency, and computational requirements.

Self-Supervised Learning for Anomaly and Drift Detection

While self-supervised learning (SSL) has shown promise in anomaly detection across domains such as image processing and financial transactions [48, 18], its application to drift detection remains underexplored. For instance, Madan et al. [48] demonstrate how SSL can effectively reconstruct masked data in high-dimensional spaces to identify anomalies. In contrast, Chen et al. [18] leverage SSL to model temporal patterns in financial transactions for fraud detection. These methods highlight SSL's ability to function without labelled data, a common constraint in drift detection scenarios.

However, the adaptation of SSL to drift detection presents unique challenges. Incorporating drift-specific evaluation metrics, such as changes in $P(X)$ or $P(Y|X)$, requires additional algorithmic modifications. Furthermore, SSL methods often involve computationally intensive processes, such as transformer-based architectures or embedding-based sampling, which may hinder their practicality for real-time drift detection in resource-constrained environments. These limitations parallel those encountered with unsupervised learning approaches, making SSL an area of potential but underutilised application in drift detection research.

Ensemble-Based Drift Detection Methods

Ensemble methods leverage multiple models to mitigate the impact of drift, often combining supervised and unsupervised approaches. Dynamic Weighted Majority (DWM) adjusts classifier weights based on performance, introducing new models when ensemble accuracy drops. While robust to various drift types, it demands significant computational resources [41]. The Accuracy Updated Ensemble (AUE) enables continuous updating of classifiers, effectively addressing abrupt and gradual drift, but risks losing model diversity over time [12, 11]. Methods like Learn++.NSE and Learn++.NIE employ incremental learning and dynamic weighting to adapt to drift but may retain outdated models, potentially degrading performance [57, 52].

Motivation for a New Drift Detection Framework

A new drift detection framework is necessary beyond standard supervised and unsupervised methods, particularly in complex systems such as *Large Language Models (LLMs)* and evolving data environments, due to several inherent challenges and limitations of traditional approaches. Standard drift detection techniques rely on explicit ground truth labels (supervised) or statistical comparisons of input distributions (unsupervised). However, these methods often fall short when applied to high-dimensional, continuously evolving data streams, where obtaining labels

is expensive and feature distributions are dynamic and complex. Several key factors identified in the analysed approaches motivate the need for an improved framework.

1. **High Dimensionality and Complex Representations.** LLMs and other deep learning models encode vast amounts of information into embeddings and weight distributions, making traditional drift detection techniques insufficient. Unlike simple statistical measures, changes in latent representations can be subtle and require specialized techniques such as *embedding shift monitoring* and *weight distribution analysis*, which track internal model behavior rather than relying solely on input-output changes. Existing frameworks do not fully exploit these internal representations, leading to delayed or incomplete drift detection [20].

   Furthermore, embedding-based drift detection offers increased robustness to noise compared to raw feature-based methods:

   - **Semantic Abstraction.** Embeddings capture high-level semantic patterns, effectively filtering out low-level noise such as typos in text, sensor jitter, or UI layout changes. This abstraction allows models to focus on meaningful patterns rather than superficial differences [61].
   - **Reduced Sensitivity to Minor Perturbations.** Raw features treat each input dimension independently, making models sensitive to minor perturbations like pixel flips or word swaps, which can lead to significant shifts in feature space and trigger false positives in drift detection [64].
   - **Compression and Denoising.** When input data is processed through models like BERT or CNNs, it is transformed into embeddings—compact vector representations where semantically similar inputs are closer together, even if their raw features differ. This compression reduces sensitivity to irrelevant variations [10].
   - **Generalisation Through Training Techniques.** Deep learning models are often trained with techniques like regularisation, dropout, and data augmentation, which encourage them to generalise beyond noisy inputs. As a result, their embeddings inherently filter out fluctuations that don't affect the label [10].

2. **Label Scarcity and Cost Efficiency.** Supervised drift detection methods require continuous access to ground truth labels, often impractical in real-world applications involving LLMs, such as chatbots and document processing. As the literature discusses, meta-learning-driven approaches and uncertainty-based drift detection provide adaptive mechanisms that minimise labelling costs by dynamically adjusting query strategies. A new framework should incorporate adaptive sampling techniques to optimise the trade-off between detection accuracy and labelling effort [24, 58].

3. **Model Interpretability and Explainability.** Traditional unsupervised drift detection methods, such as statistical distance measures, provide little insight into *why* drift occurs within LLMs. Methods such as *activation-based uncertainty estimation* give a more profound understanding by quantifying uncertainty across different model layers. A novel drift detection framework should focus on improving interpretability, offering actionable insights that enable targeted retraining rather than indiscriminate model updates [20].

4. **Easier Integration with MLOps Pipelines.** For LLMs deployed in production environments, drift detection frameworks should seamlessly integrate with existing MLOps pipelines to trigger automated model retraining and updates. Unlike conventional frameworks, which require manual intervention, advanced frameworks should enable *real-time detection and self-adaptive correction*, ensuring minimal disruption to business operations [16].

While modern approaches such as embedding-based and self-supervised methods offer improved flexibility, they still rely heavily on detecting statistical shifts in feature space or model outputs, often assuming that any significant change signals a meaningful drift event. However, recent findings challenge this assumption. Chen et al. [20] show that performance degradation in practical malware detection systems is primarily driven by changes in data-space distributions, not by newly introduced features or changes in the embedding space. In other words, many traditional and modern drift detectors may trigger adaptation based on statistically significant but functionally irrelevant shifts. A more nuanced framework is needed to evaluate whether a shift has occurred and whether that shift affects model performance or semantic understanding [45, 48]. The following section addresses this gap by proposing a model-integrated approach that monitors the internal states of deep learning models for meaningful concept drift.

2.3 Deep Learning for Concept Drift Detection

Embedding-based drift detection methods operate under the premise that if the underlying distribution of data changes over time, this shift will manifest in the learned internal representations of that data. These embeddings, typically learned via autoencoders, deep neural encoders, or language models, offer a dense semantic view of the data, making them more sensitive to distributional nuances than raw feature-based approaches. For example, in the domain of textual anomaly detection, models like BERT and MiniLM are used to encode text streams into latent vectors, after which unsupervised detectors such as k-nearest neighbours or isolation forests can be applied to identify anomalous segments potentially indicating drift [13].

The advantage of using embeddings lies in their ability to capture non-linear relationships and abstract patterns in the data, which are often invisible in the original feature space. Euclidean distance and cosine similarity are standard metrics for measuring divergence between embeddings across time windows. A consistent increase in the distance between mean

embeddings of temporally segmented data often signals a change in the underlying concept [69]. Maximum Concept Discrepancy (MCD), introduced in recent work, formalises this approach by applying contrastive learning to optimise embeddings for separability across time, using dynamic thresholds derived from empirical and theoretical bounds to flag drifts [69].

One advantage of deep representation monitoring is its generality. It can be applied across modalities, text, time series, and sensor data, and does not rely on domain-specific assumptions. Moreover, embedding-based approaches can be entirely unsupervised, a critical requirement in real-world scenarios where labelled data is scarce or unavailable. In these contexts, models are often trained using self-supervised contrastive learning objectives, such as InfoNCE, which pull together similar representations and push apart temporally distant or augmented ones [69]. The resulting latent space becomes structured to reflect the temporal evolution of concepts, allowing drift to be detected as geometric shifts.

However, embedding-based methods also face notable challenges. One of the primary issues is the computational cost associated with training and updating deep encoders in real-time or near-real-time environments. Although recent work has introduced optimisations for online adaptation, including sliding window encoders and lightweight sampling strategies [69], these models remain more resource-intensive than traditional statistical detectors. Another challenge is interpretability. While embeddings can be visualised using techniques like t-SNE, the semantic meaning of detected drifts is often opaque, especially in high-dimensional settings. Recent work attempts to mitigate this by combining embeddings with domain-specific structures, such as knowledge graphs [19], or by incorporating explanation modules that localise changes within the latent space [36].

Despite these challenges, recent benchmarks suggest that embedding-based detectors are becoming competitive with, and in some cases superior to, traditional approaches. Wan et al. [69] demonstrated that their MCD-DD method outperformed classical detectors like Kolmogorov-Smirnov and Maximum Mean Discrepancy with Gaussian kernels across synthetic and real-world data sets in precision, F1-score, and MCC. Moreover, the method provided qualitative insights into the nature of drift by mapping concept representations in two-dimensional spaces, revealing distinct patterns for sudden, incremental, and recurring drifts.

Chapter 3

Mathematical Foundations of Drift Detection

3.1 Vector/Feature-Space Distances in Drift Detection

In recent work, feature-space or vector-based distance methods have gained traction as robust detectors of shifts in data diffusions. By transforming raw observations into dense vectors or more abstract features, these approaches rely on computing pairwise distances to look for variations in magnitude or direction that signal potential drift. [53] shows the importance of monitoring these feature distances when performing tasks like robust object tracking, where apparent drift in the correlation response can be interpreted as a broader shift in the underlying data distribution. Meanwhile, Barboza et al. proves that data normalisation policies and careful choice of distance metrics (e.g., Canberra, Minkowski) significantly influence the reliability of drift detection, highlighting how small changes in data ranges can either conceal or amplify underlying changes. These findings highlight the need for carefully chosen feature representations and distances in scenarios where data evolve continuously and unpredictably.

While scalar comparisons are sufficient for stable data, the dynamic nature of streaming contexts calls for distance measures that adapt seamlessly to changing patterns. As [56] explains, embedding-based techniques for time-series and streaming data provide a structured means of mapping high-dimensional signals into latent spaces, often applying Euclidean or cosine distances post-embedding to gauge novelty or anomaly. When these embedding spaces are suitably learned or handcrafted, moderate increases in distance may indicate distributional shifts, prompting the timely reconfiguration of models. Aligning these concepts with the robust detection methods explored in [53] and the nuanced normalisation frameworks from [6], it becomes clear that feature-space distances can detect gross changes and subtle deviations that might otherwise go unnoticed. These embeddings, distance metrics, and careful normalisation stand at the forefront of contemporary drift detection research, enabling more proactive and flexible responses to ever-evolving data.

This section explores how feature-space distances unite robust representation learning with real-time drift detection in dynamic data streams. First, it details foundational distance metrics and embedding spaces, examining how Euclidean versus cosine can highlight different

facets of evolving data. Next, it discusses the practical implications of normalisation (as outlined in [6]) and computational cost, offering insights into how each factor may mask or reveal subtle distributional changes. Finally, it explores strategies for implementing these methods into adaptive models, highlighting the proactive response needed in environments prone to abrupt or incremental shifts. The section weaves together the choice of embeddings, distance metrics, and normalisation frameworks, enabling a more resilient and fine-grained drift detection pipeline.

3.1.1 Overview of Vector/Feature-Space Distances Metrics

Understanding how different distance metrics behave in high-dimensional or evolving feature spaces is critical for effective drift detection. Each metric offers unique properties that make it more or less suitable, depending on the nature of the data: whether it is dense or sparse, continuous or categorical, and whether it is static or streaming. In adaptive learning systems, having a principled choice of distance measure can make the difference between early detection and missed drift. Table 3.1 summarises key metrics commonly used for measuring feature-space divergence, their defining formulas, behavioural traits, and sensitivity characteristics.

Table 3.1: Overview of vector/feature-space distance metrics and their properties.

Metric	Formula	Key Characteristics	Sensitivity
Euclidean	$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$	Straight-line (L2 norm); emphasizes magnitude	Sensitive to scale and outliers
Manhattan (L1)	$\sum_{i=1}^{n} \lvert x_i - y_i \rvert$	Sum of absolute differences; axis-aligned	Less sensitive to outliers; scale-dependent
Minkowski	$\left(\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p\right)^{1/p}$	Generalized form of L1 and L2 (with tunable p)	Sensitivity depends on p
Chebyshev	$\max_i \lvert x_i - y_i \rvert$	Maximum coordinate deviation	Very sensitive to single-feature changes
Mahalanobis	$\sqrt{(x-y)^T S^{-1}(x-y)}$	Scale- and correlation-aware via covariance matrix	Sensitive to poor covariance estimation
Canberra	$\sum_{i=1}^{n} \frac{\lvert x_i - y_i \rvert}{\lvert x_i \rvert + \lvert y_i \rvert}$	Strong sensitivity to small relative changes	Sensitive to noise near zero
Cosine	$1 - \frac{x \cdot y}{\lVert x \rVert \lVert y \rVert}$	Captures angular difference; ignores magnitude	Insensitive to scale; sensitive to direction

It is helpful to visualise how these metrics respond to asymmetries in the data. In the following examples, two vectors $\mathbf{X} = [1, 10]$ and $\mathbf{Y} = [10, 11]$ have comparable overall magnitudes but differ sharply along one axis. This contrast allows us to observe how each metric interprets spatial displacement, scale, feature dominance, and relational geometry differently, highlighting their diverse sensitivity profiles.

Euclidean (L2) Distance:.

$$d_{\text{Euclidean}}(\mathbf{X}, \mathbf{Y}) = \sqrt{(1-10)^2 + (10-11)^2} = \sqrt{(-9)^2 + (-1)^2} = \sqrt{81+1} = \sqrt{82} \approx 9.055.$$

The Euclidean distance accentuates the magnitude of differences, capturing both dimensions but being more strongly influenced by the larger horizontal gap (9) than the smaller vertical gap (1). It is geometrically faithful because it measures actual geometric distance and is well-suited for continuous and spatial data representations [22]. Training on samples with higher Euclidean distances between embeddings in NLP tasks has resulted in better generalisation and robustness

in language models [3]. In image processing, it is more suitable than city-block or octagonal metrics for constructing accurate distance maps [22], and distance errors decrease significantly with increasing distance, making it reliable for detecting larger-scale drifts [22]. However, Large feature differences can disproportionately influence Euclidean distance, which is more computationally intensive than Manhattan distance for large grids or high-dimensional data [66], and may produce minor approximation errors in discrete image grids when using specific sequential algorithms [22].

Manhattan (L1) Distance:.

$$d_{\text{Manhattan}}(\mathbf{X}, \mathbf{Y}) = |1 - 10| + |10 - 11| = 9 + 1 = 10.$$

The Manhattan distance, computed by summing absolute differences along each axis, is relatively large here because each coordinate difference contributes additively. It is easier to calculate than Euclidean distance, making it well-suited for clustering tasks (e.g., K-Means, KNN) on structured or tabular data [66]. It is also more robust to outliers than L2 norms because it is based on absolute differences and can require fewer iterations in clustering tasks [66]. On the other hand, Manhattan distance can produce blocky, diamond-shaped boundaries in distance maps, which may distort spatial interpretations [22]. It also performs poorly in capturing semantic differences in high-dimensional vector spaces (e.g., word embeddings). It also shows inferior accuracy to other metrics in specific image and satellite data classification tasks [66].

Minkowski Distance (general p):.

$$d^p_{\text{Minkowski}}(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} |x_i - y_i|^p, \quad d_{\text{Minkowski}}(\mathbf{X}, \mathbf{Y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}.$$

For $p = 1$, it recovers Manhattan; for $p = 2$, it recovers Euclidean. At $p = 3$ for the vectors above, the distance is approximately 9.006, showing how higher p values amplify more significant coordinate gaps. The Minkowski distance can be very flexible in adapting to the nature of the data, but for $p < 1$, it is no longer a proper metric because it violates the triangle inequality [60].

Chebyshev Distance (L^∞):.

$$d_{\text{Chebyshev}}(\mathbf{X}, \mathbf{Y}) = \max\left(|1 - 10|,\ |10 - 11|\right) = \max(9,\ 1) = 9.$$

This distance focuses on the maximum coordinate difference and discards smaller gaps, making it highly efficient for neighbourhood iteration in specific 2D spaces [60]. Its simplicity eases some implementations but can be associated with weaker performance for many tasks [60].

Mahalanobis Distance:.

$$d_{\text{Mahalanobis}}(\mathbf{X}, \mathbf{Y}) = \sqrt{(\mathbf{X} - \mathbf{Y})^{\mathsf{T}} S^{-1} (\mathbf{X} - \mathbf{Y})},$$

where S is the covariance matrix. Suppose

$$S = \begin{pmatrix} 2 & 0 \\ 0 & 50 \end{pmatrix}, \quad S^{-1} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{50} \end{pmatrix}.$$

Then

$$(\mathbf{X} - \mathbf{Y}) = [-9,\ -1], \quad (\mathbf{X} - \mathbf{Y})^{\mathsf{T}} S^{-1} = [-4.5,\ -0.02]\,,$$

$$d_{\text{Mahalanobis}}(\mathbf{X}, \mathbf{Y}) = \sqrt{[-4.5,\ -0.02] \cdot [-9,\ -1]} = \sqrt{40.5 + 0.02} = \sqrt{40.52} \approx 6.368.$$

Mahalanobis distance incorporates the covariance structure of the dataset, making it practical for identifying outliers in multivariate data [31, 23]. It is scale-invariant and shapes the distance metric according to data covariance, helping it measure how far an observation lies from the mean in a correlated feature space [23]. Robust estimators, such as the Minimum Covariance Determinant, can mitigate issues of masking effects when outliers inflate variances [31, 23]. Mahalanobis distance is also the basis for several learning algorithms that rely on task-specific metric learning [73], with uses across chemometrics, pattern recognition, image segmentation, face pose estimation, and fraud detection [73, 31]. However, it is susceptible to outliers in its classic form because it depends on the sample mean and covariance [23], and computing the inverse covariance matrix can be expensive in high-dimensional data [31, 23]. It also requires that the number of samples exceed the number of variables, and it can suffer if features are highly correlated or if the data deviate from a multivariate normal distribution [23, 31].

Canberra Distance:.

$$d_{\text{Canberra}}(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}.$$

For the 2D example, this sums to approximately 0.8657. Canberra distance is sensitive to relative changes rather than absolute magnitudes, so small values in a feature can produce large fractional swings [26]. This sensitivity can be instrumental when subtle changes in lower-valued features are crucial for detecting distributional drift.

Why These Metrics Matter for Embedding Drift Detection?

These distance metrics take on special significance in embedding-based drift detection because the nature of the embedding space—whether sparse or dense, directional or magnitude-sensitive can determine which distance function most effectively uncovers drift. Scale sensitivity is one consideration, since Euclidean, Minkowski, and Manhattan distances can overemphasise large-range features, whereas Mahalanobis accounts for covariance and thus recognises correlations

in high-dimensional embeddings. Sparsity versus density also matters, since L1-based metrics like Manhattan handle sparse embeddings differently than L2-based measures. Cosine distance, focusing on angular difference, has become a common choice in text or embedding spaces precisely because it is insensitive to scale and captures directional changes. Relative versus absolute changes come into play when considering Canberra's ratio-based formulation, which excels for small-valued features, or Chebyshev's reliance on a single feature's maximum deviation. By carefully selecting a distance measure that aligns with the embedding space and the nature of potential drift, practitioners can more reliably detect when an embedding has drifted, thereby prompting model recalibration, data inspection, or adaptation steps.

3.2 Distribution-Based Distances and Divergences

Unlike feature-space distances (Section 3.1), which focus on pointwise comparisons between vectors in embedding spaces between points $f(x, y)$, distribution-based metrics gauge the divergence or distance between two probability distributions as a whole $P(p|q)$. These metrics help detect drift when it is possible to estimate distributions from samples or explicit density models and assess how those distributions change over time. In streaming contexts, distributional changes can signal shifts that purely local or pointwise measures might otherwise miss. Therefore, these distances serve as global descriptors of how probability mass is reallocated over outcomes, offering an expanded view of potential changes in data over time.

Distribution-based distances are indispensable for quantifying changes between probability distributions, particularly in drift detection and generalisation assessment in machine learning systems. Unlike pointwise measures, which compare individual data points or feature embeddings, these metrics characterise entire distributions and allow practitioners to assess whether two datasets stem from the same generating process. Such metrics are widely employed in domains including covariate drift detection, simulation validation, and transfer learning, where assumptions about the stationarity or similarity of distributions directly affect the reliability of downstream models [17, 1, 65].

In drift detection, distributional distances are statistical probes to identify shifts in the input features (covariate drift) or the learned target relationships (concept drift). These changes are typically revealed by comparing empirical distributions over time or across different domains. Two-sample tests, particularly those built upon divergence measures such as the Kullback–Leibler divergence, Jensen–Shannon divergence, Hellinger distance, or Wasserstein metrics, are central to this analysis [1]. In systems with streaming data or limited labels, these measures enable unsupervised drift monitoring by comparing feature distributions using kernel-based, rank-based, or graph-based techniques. For example, serverless ML pipelines can incorporate tools like the Frouros library to compute multiple such divergences in batch mode, enhancing their robustness to real-world nonstationary data [17].

3.2.1 Overview of Distribution-Based Distances and Divergences

Table 3.2: Overview of distribution-based metrics and their properties.

Metric	Formula (Discrete Case)	Key Characteristics	Sensitivity
Kullback-Leibler (KL) Divergence	$D_{\mathrm{KL}}(P \parallel Q) = \sum_i p_i \log\left(\frac{p_i}{q_i}\right)$	Non-symmetric; info loss from Q to P	Sensitive when $q_i \approx 0$, $p_i > 0$
Jensen–Shannon (JS) Divergence	$D_{\mathrm{JS}}(P \parallel Q) = \frac{1}{2}D_{\mathrm{KL}}(P \parallel M) + \frac{1}{2}D_{\mathrm{KL}}(Q \parallel M)$	Symmetric; bounded and stable	Less sensitive than KL; handles mismatched support
Hellinger Distance	$d_{\mathrm{H}}(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_i(\sqrt{p_i} - \sqrt{q_i})^2}$	Overlap-focused; bounded in $[0, 1]$	Sensitive to differences in small probabilities
Bhattacharyya Distance	$d_{\mathrm{B}}(P,Q) = -\ln \sum_i \sqrt{p_i\, q_i}$	Measures overlap; non-negative	Sensitive to low-support overlap
Wasserstein (1D)	$W_1(P,Q) = \int \lvert F_P(t) - F_Q(t)\rvert \, dt$	Earth mover's distance; interprets mass shift	Sensitive to distributional shifts in shape/location
Maximum Mean Discrepancy (MMD)	$\mathrm{MMD}(P,Q) = \left\lVert \frac{1}{n}\sum \phi(x_i) - \frac{1}{m}\sum \phi(y_j)\right\rVert_{\mathcal{H}}$	Kernel-based mean difference	Kernel-dependent; captures subtle shifts

KL Divergence. measures the expected log-likelihood ratio between two probability distributions and is defined as

$$D_{\mathrm{KL}}(P\|Q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx.$$

It quantifies how much information is lost when Q is used to approximate P, and is widely used in information theory, statistics, and machine learning. However, KL divergence is asymmetric and can diverge to infinity if P assigns mass where Q does not (i.e., when $P \not\ll Q$), which limits its robustness in some applications. Despite this, it satisfies key properties such as non-negativity, convexity in (P, Q), and the data processing inequality, making it a foundational tool for measuring information discrepancy [25]. KL divergence is additive under independent or Markovian models, enabling tractable analysis of sequences and structured distributions [38]. While not a true metric, it can be symmetrised using constructs like the J-divergence or the resistor-average divergence, which aim to balance interpretability, symmetry, and performance-related bounds in classification and estimation tasks [38].

JS Divergence. could be moderate if P's support does not vanish where Q is nonzero; since distributions overlap, $D_{\mathrm{KL}}(P\|Q)$ should remain finite but is asymmetric (i.e., $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$). The *Jensen–Shannon divergence* (JSD) is often preferred to address this asymmetry and potential divergence when supports differ. Defined as a symmetrised and smoothed version of KL divergence:

$$\mathrm{JSD}(P,Q) = \frac{1}{2}D_{\mathrm{KL}}\left(P\|M\right) + \frac{1}{2}D_{\mathrm{KL}}\left(Q\|M\right), \quad \text{where } M = \frac{1}{2}(P+Q),$$

JSD is always bounded (by $\log 2$ for normalised distributions), symmetric, and defined even when P and Q have disjoint supports. Moreover, the square root of JSD, $\sqrt{\mathrm{JSD}}$, defines a proper metric and admits an isometric embedding into Hilbert space, enabling geometric interpretations and kernel-based learning algorithms [28]. JSD can also be expressed as the difference between the entropy of the mixture and the average entropy:

$$\mathrm{JSD}(P,Q) = H\left(\frac{P+Q}{2}\right) - \frac{1}{2}\left(H(P) + H(Q)\right).$$

Extensions such as the α-skew and vector-skew Jensen–Shannon divergences introduce tunable parameters to adapt divergence sensitivity [54]. These generalisations preserve convexity and boundedness, allowing for fine-grained control in applications such as clustering, natural language processing, and hypothesis testing [51].

Bhattacharyya Distance. is a symmetric, bounded measure of similarity between two probability distributions, defined as

$$B(P, Q) = -\ln\left(\int \sqrt{p(x)q(x)}\, dx\right).$$

It corresponds to the Chernoff distance with exponent $s = 1/2$ and is particularly valued in applications such as signal selection, pattern recognition, and image analysis in noisy environments. Compared to KL divergence, the Bhattacharyya distance often provides more reliable approximations to classification error probabilities and is computationally more tractable than the full Chernoff distance [39]. It is convex in the likelihood ratio and maintains key properties under affine transformations, although it does not satisfy the triangle inequality and is therefore not a true metric. In practical tasks such as image segmentation under non-Gaussian noise (e.g., speckle or Poisson), the Bhattacharyya distance has been shown to serve as an effective scalar contrast parameter, aligning well with empirical performance across varying noise models [33].

Hellinger Distance. is a true metric between probability distributions, defined as

$$H(P, Q) = \frac{1}{\sqrt{2}}\left(\sum_x \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2\right)^{1/2}.$$

It is symmetric, bounded in $[0, 1]$, and invariant under bijective transformations, making it suitable for non-parametric distribution comparison problems. As a member of the f-divergence family, it is particularly effective in detecting distributional shifts. It has found widespread use in quantification problems, such as estimating class priors when the test distribution differs from the training one [32]. Unlike KL divergence, Hellinger distance does not require absolute continuity (i.e., overlapping support), making it robust in domains with covariate shift or imbalanced labels. Practical methods such as HDx and HDy leverage Hellinger distance on feature and classifier output distributions, respectively, and consistently outperform naive estimators in class distribution estimation across real-world domains.

Wasserstein (1D). accounts for the "movement" of probability mass among the three outcomes, effectively computing the integral of the absolute differences between cumulative distribution functions. The 1-Wasserstein distance, also known as the Earth Mover's Distance, is defined as

$$W_1(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|],$$

where $\Pi(P, Q)$ denotes the set of all couplings of P and Q. Unlike f-divergences such as KL or JS, W_1 remains finite and meaningful even when the supports of the distributions do not overlap. It captures the geometric structure of the probability space and is particularly suited for tasks involving distributions supported on low-dimensional manifolds [21]. The 1-Wasserstein distance exhibits a favourable optimisation landscape: in contrast to mode-seeking divergences that often lead to mode collapse, gradient descent on W_1 avoids poor local minima and tends to recover all modes of a multi-modal target distribution [44]. This property has made it a powerful objective for generative models such as Wasserstein GANs. Furthermore, its dual form involving 1-Lipschitz functions makes it robust to vanishing gradients, contributing to the stability and convergence of training [72].

Maximum Mean Discrepancy (MMD). is a kernel-based, nonparametric divergence that measures the distance between two probability distributions based on the difference of their mean embeddings in a reproducing kernel Hilbert space (RKHS). Formally, it is defined as

$$\mathrm{MMD}(P, Q) = \sup_{f \in \mathcal{H}, \|f\|_{\mathcal{H}} \leq 1} \left(\mathbb{E}_{x \sim P}[f(x)] - \mathbb{E}_{y \sim Q}[f(y)] \right),$$

where $\mathcal{H}$ is a unit ball in the RKHS induced by a universal kernel. MMD has been widely adopted in two-sample testing, especially for detecting concept drift in high-dimensional or non-Gaussian data streams. Unlike moment-based statistics, it captures higher-order differences between distributions and performs well even when mean or variance remain constant [60]. In practice, MMD avoids explicit function optimisation by expressing the distance in pairwise kernel evaluations, making it computationally efficient. More recently, MMD has inspired adaptive methods such as Maximum Concept Discrepancy (MCD), which replaces fixed kernels with deep encoders to learn data-driven embeddings for unsupervised drift detection in streaming settings [69]. These extensions enable dynamic thresholding, contrastive training, and online updates, allowing for more accurate and interpretable detection of both abrupt and gradual distribution shifts.

Choosing the Right Metric for Distributional Drift Detection

When diagnosing drift using distribution comparisons, the metric choice can impact the reliability of the results. In settings where two distributions have partially disjoint supports, metrics such as the KL divergence can become unbounded if one distribution assigns a negligible probability to events the other considers likely. Consequently, Jensen–Shannon, Hellinger, or Bhattacharyya measures often provide more stable behaviour when distributions do not overlap perfectly. If the application requires symmetry, those measures (rather than KL) are again more appropriate. In continuous or ordinal domains, the Wasserstein distance excels by quantifying the "effort" of moving probability mass in the underlying metric space. Meanwhile, Maximum Mean Discrepancy (MMD) can reveal subtle discrepancies by leveraging kernel functions and

highlighting domain-relevant features. In practice, the chosen metric should align with the characteristics of the data, whether sparse, dense, discrete, or continuous, and with the expected form of drift, such as shifts in distribution shape, location, or tail heaviness. Approximations like histograms and KLL sketches often make using these metrics at scale or in streaming scenarios feasible. Still, they can affect how finely the probability mass is resolved and thus the sensitivity of drift detection.

3.3 A Comparative Analysis PCA and the KLL Sketch Algorithm

This section compares two approaches to summarising high-dimensional data: Principal Component Analysis (PCA) and the KLL sketch algorithm. While PCA captures global structure by projecting data onto lower-dimensional subspaces that preserve variance, KLL sketches approximate univariate quantile information in a streaming fashion with strict space guarantees. We examine the mathematical foundations, computational trade-offs, and memory requirements of each method to highlight their complementary strengths and limitations.

3.3.1 Principal Component Analysis (PCA)

Mathematical Foundation

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that identifies directions with the highest variance in a dataset [4, 59]. Consider a data matrix $\mathbf{X}$ of size $n \times d$, where each of the n rows is a data point and each of the d columns is a feature or dimension. A typical approach begins by computing the $d \times d$ covariance matrix,

$$\mathbf{\Sigma} = \frac{1}{n}\mathbf{X}^\top\mathbf{X}.$$

Once the covariance matrix has been formed, an eigen decomposition or singular value decomposition (SVD) is performed. In the SVD view, we express $\mathbf{X}$ as

$$\mathbf{X} = \mathbf{U}\,\mathbf{S}\,\mathbf{V}^\top,$$

where $\mathbf{V}$ contains right singular vectors that correspond to the eigenvectors of $\mathbf{\Sigma}$. By ordering the singular values in descending order, we identify the principal components that capture the greatest variance in the data. To reduce dimensionality from d to k, the data points are projected onto the first k columns of $\mathbf{V}$. If $\mathbf{V}_k$ represents this selection of eigenvectors, each point $\mathbf{x}_i$ is mapped to

$$\widehat{\mathbf{x}}_i = \mathbf{x}_i\,\mathbf{V}_k,$$

thus preserving the bulk of the variance in just k dimensions.

Algorithmic Complexity and Memory

Given a data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, computing the covariance matrix $\mathbf{\Sigma} = \frac{1}{n}\mathbf{X}^\top \mathbf{X}$ requires $O(nd^2)$ operations when d is large. This step can be a significant computational bottleneck in high-dimensional settings [67]. Once the covariance matrix is computed, performing eigenvalue decomposition on the $d \times d$ matrix $\mathbf{\Sigma}$ has a computational complexity of $O(d^3)$. This step is necessary to obtain the principal components. Truncated Singular Value Decomposition (SVD) can mitigate computational costs, focusing on the top k components. This approach reduces the complexity to approximately $O(ndk)$, which is more efficient when $k \ll d$. Storing the full covariance matrix demands $O(d^2)$ memory, which becomes impractical as d increases [59].

Additionally, retaining all principal components requires $O(dk)$ memory. These requirements pose challenges in high-dimensional scenarios. To handle large datasets that cannot fit into memory, incremental PCA algorithms process data in batches, updating the principal components iteratively. These methods maintain a memory footprint that is independent of the number of samples n, although they still scale with the number of features d [14]. Randomised algorithms use random matrices to approximate the principal components by projecting the data onto a lower-dimensional subspace. This technique significantly reduces computational time and memory usage, making it suitable for large-scale applications. While approximate methods alleviate some computational burdens, achieving truly sublinear time or memory complexity in PCA is challenging. This difficulty arises because capturing the variance structure of the data inherently requires processing a substantial portion of the dataset.

Numerical and Implementation Details for Streaming PCA

While classical PCA methods form and decompose a full covariance matrix, streaming or incremental PCA algorithms operate on data points one at a time (or in small batches) and never construct the entire $\mathbf{\Sigma}$. Instead, they maintain a low-rank factorisation or subspace representation as new samples arrive [4]. One common approach, sometimes called truncated incremental SVD, keeps a rank-k approximation $\mathbf{C}^{(t-1)}$ of the sample covariance. On arrival of a new vector $\mathbf{x}_t$, it forms

$$\mathbf{C}^{(t)} \;=\; \text{Prank-}k\big(\mathbf{C}^{(t-1)} + \mathbf{x}_t \mathbf{x}_t^\top\big),$$

where Prank-k projects onto the top k eigencomponents. In practice, an eigendecomposition $\mathbf{U}\,\mathbf{S}\,\mathbf{U}^\top$ of $\mathbf{C}^{(t-1)}$ is updated with a rank-1 correction $\mathbf{x}_t \mathbf{x}_t^\top$. Each incremental step typically costs $O(k^2 d)$, which is smaller than forming and factoring a full $d \times d$ covariance, and storing $\mathbf{U}$ and $\mathbf{S}$ in $O(kd)$ space is more memory-friendly than $O(d^2)$ [4].

Another well-known approach is the stochastic power method, which processes each new sample by

$$\mathbf{U}^{(t+1)} = \mathbf{U}^{(t)} \;+\; \eta_t \, \mathbf{x}_t \, \mathbf{x}_t^\top \, \mathbf{U}^{(t)},$$

and optionally applies a normalisation or Gram-Schmidt step. A step size $\eta_t = 1/\sqrt{t}$ is often used, each iteration costs roughly $O(kd)$, and less frequent orthonormalization prevents numerical drift at a cost of $O(k^2 d)$ if done too often. Stochastic methods may converge more slowly or require tuning η_t, whereas incremental SVD can occasionally converge to the wrong subspace in pathological distributions. Nevertheless, both methods avoid the $O(nd^2)$ overhead of batch PCA and tend to perform well in practice [4]. Theoretical rates for truncated incremental SVD remain an open question; while more expensive, some exponentiated-gradient approaches enjoy online regret bounds. A single pass over the data can often yield near-batch quality for moderate k, though multiple passes or a buffered replay may further refine the learned subspace [4].

3.3.2 KLL Sketch

Purpose and Guarantees

The KLL sketch is a compact data structure designed by Karnin, Lang, and Liberty to approximate quantiles in data streams. For a sequence of items $\{x_1, x_2, \ldots, x_n\}$, the sketch returns an approximate rank $\tilde{R}(x)$ for any query value x, where the rank $R(x)$ denotes the number of items in the stream less than or equal to x. Its core guarantee is that

$$\left|\tilde{R}(x) - R(x)\right| \leq \varepsilon n$$

with high probability $1-\delta$. In other words, estimating how many items in the stream are less than or equal to x is accurate up to εn. Figure 3.1 offers a simple depiction of this rank-approximation idea.

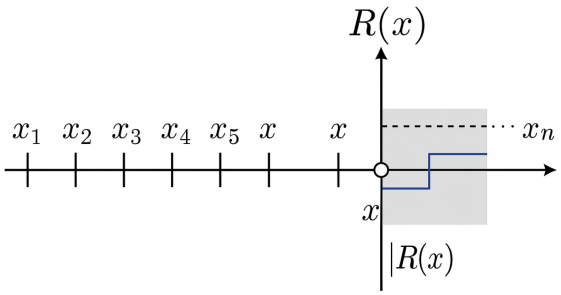


Figure 3.1: Illustration of the approximate rank guarantee in KLL Sketch

Core Data Structure

KLL relies on a hierarchical collection of *compactors*. Each compactor has a capacity and holds items until the compactor is full. When the capacity is reached, the compactor sorts its items, randomly discards half of them (either at odd or even indices), doubles the weight of the retained items (interpreted as each representing two discarded items), and sends them to a

higher-level compactor. This way, levels deeper in the hierarchy represent exponentially larger subsets of the original stream. The capacity at each level typically shrinks by a constant factor as we move upward in the hierarchy, ensuring that memory usage remains bounded. Figure 3.2 shows a simplified view of multiple compactors at different levels, with decreasing height from H down to 1.

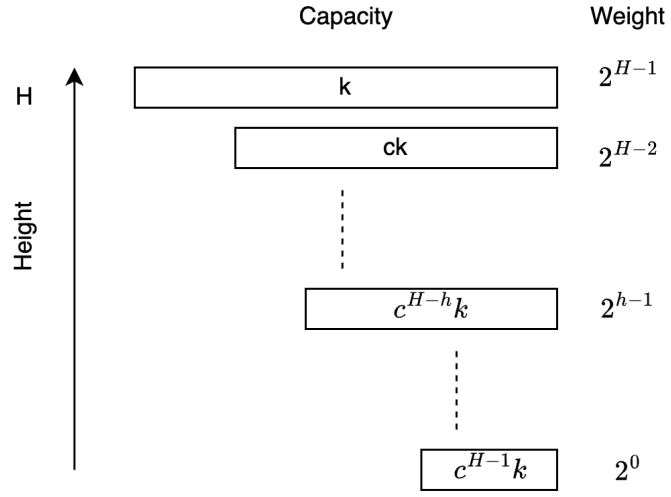


Figure 3.2: A series of compactors arranged in levels, each with capacity and weight assignment.

Randomisation ensures that the compaction process introduces only minor expected errors. According to Hoeffding's inequality, the cumulative rank error remains bounded with high probability. Figure 3.3 shows how discarded or retained items may introduce positive or negative rank errors, yet the average outcome remains unbiased [74].

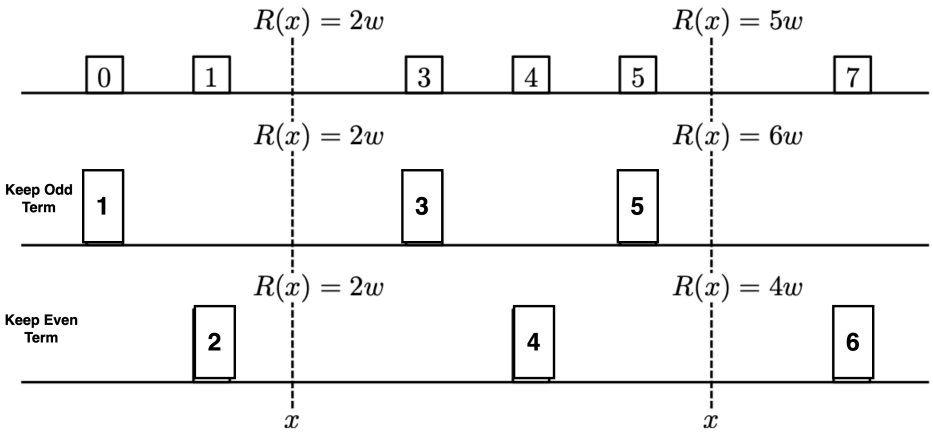


Figure 3.3: Compaction step showing how some items are discarded while others double in weight and move upward.

The mergeable property of the KLL sketch allows two sketches, built with identical parameters, to be combined by merging their items at each level. This feature is particularly valuable in distributed or parallel processing contexts. Additionally, a non-mergeable variant

that places a Greenwald-Khanna (GK) sketch at the top level can reduce space further when mergeability is not required. In either form, the space complexity for storing a KLL sketch is independent of the stream length n, and depends primarily on ε, δ, and logarithmic factors. The space cost is per dimension, so summarising quantiles across multiple dimensions grows linearly with the number of dimensions.

Space Complexity

A mergeable KLL sketch generally requires on the order of

$$\mathcal{O}\Big(\frac{1}{\varepsilon}\ \log^2\!\log\frac{1}{\delta}\Big)$$

space, while substituting a GK sketch at the top level, if mergeability is not needed, can reduce this to

$$\mathcal{O}\Big(\frac{1}{\varepsilon}\ \mathrm{loglog}\frac{1}{\delta}\Big).$$

Because each dataset dimension can be sketched separately, the total cost across dimensions grows linearly, while remaining sublinear in the stream length n.

3.3.3 Fundamental Differences between PCA and KLL Sketch

Principal Component Analysis (PCA) seeks a *global* linear subspace that captures variance across all d dimensions. Consequently, it must store covariance information or principal components in a form proportional to d. In contrast, the KLL sketch handles *distributional queries* (ranks, quantiles) in each dimension separately. Maintaining a KLL sketch for one dimension consumes space on the order of

$$\mathcal{O}\Big(\tfrac{1}{\varepsilon}\ \log^2\!\log\tfrac{1}{\delta}\Big)\quad\text{or even}\quad\mathcal{O}\Big(\tfrac{1}{\varepsilon}\ \mathrm{loglog}\tfrac{1}{\delta}\Big)$$

if a non-mergeable variant is employed. The total memory grows *linearly* with the number of dimensions but remains independent of the stream length n. By contrast, incremental or streaming PCA still needs to track and update a partial covariance or an orthonormal basis whose size depends significantly on d.

PCA often requires computing a large matrix decomposition (e.g., an SVD of size $d \times d$) or updating a lower-dimensional basis. Even truncated or randomised PCA implementations cannot fully escape storing or approximating covariance information for d features. The KLL sketch, however, is inherently a *streaming* algorithm: it processes each incoming item, triggers local "compaction" when needed, and never revisits past data. This architecture keeps memory usage bounded by a function of ε and δ, rather than the data dimension.

Partly PCA models must be combined carefully in large-scale systems or distributed environments, requiring additional overhead for aggregating covariance matrices or merging

principal components. KLL, on the other hand, is trivially *mergeable*: two sketches built with the same parameters can be combined by merging items at each level of compaction. This property is crucial in parallel processing and high-speed stream scenarios, where data arrives at multiple sites that periodically share summaries instead of raw data. The memory use of each method is tied to what it preserves: PCA retains a global subspace that captures correlations and variance structure across all dimensions, making it useful for dimensionality reduction, visualization, or noise removal; KLL provides approximate ranks or quantiles in each dimension, enabling statistical summaries such as medians or percentiles without requiring storage of the full dataset. When the goal is to understand or query a specific dimension's distribution, such as asking, "What is the median sensor reading?", KLL is more memory-efficient. PCA, by contrast, remains the preferred approach when extracting a new, lower-dimensional coordinate system that reflects correlations among features.

While streaming PCA avoids storing a full $d \times d$ covariance matrix, it must still maintain $O(k\,d)$ space to capture a low-dimensional subspace spanning all d features. This improved requirement can still be prohibitive for sufficiently large d. In contrast, the KLL sketch stores only $O\big(\frac{1}{\varepsilon}\,\mathrm{loglog}(\frac{1}{\delta})\big)$ space per dimension, leading to total memory that grows linearly in d but remains sublinear in the length of the stream n. Moreover, PCA and KLL target fundamentally different objectives: PCA seeks a single global transform that captures correlations among features for dimensionality reduction, while KLL provides per-dimension distribution summaries such as ranks and quantiles. Streaming PCA retains valuable structure when preserving cross-feature correlations at the cost of higher memory usage. KLL, on the other hand, completely forgoes global correlation information to maintain dimension-wise sketches. If tasks need only approximate univariate statistics, KLL's design is much more memory-friendly in high-dimensional settings.

Thus, while PCA is invaluable for analysing global variance and extracting correlated components, its overhead remains large if d is huge. The KLL sketch's independence across dimensions ensures that storage requirements do not explode as d grows, merging is straightforward in distributed systems, and local compaction keeps each sketch small without sacrificing distributional accuracy. When the core task is maintaining quantile or rank estimates in a massive stream, KLL's sublinear memory usage in n and milder dependence on d make it a far more practical choice.

Chapter 4

Design & Implementation

This chapter presents the architecture, algorithms, and implementation details behind the proposed embedding-based drift detection framework at GitHub. Motivated by the increasing need for real-time monitoring of model stability in dynamic environments, we focus on lightweight, in-model approaches that require minimal additional infrastructure. Our system captures both geometric and statistical shifts in internal representations, offering a fine-grained view of distributional changes that can signal model drift. We begin by outlining the libraries and components used to construct our framework, followed by a detailed description of two complementary detection strategies, vector-based and distribution-based. Each method leverages embedding activations produced during inference, making them suitable for online and resource-constrained deployments.

4.1 Motivation

Building on prior research in drift detection (see Section 2), we choose an embedding-based approach for several reasons. First, embeddings provide a unifying representation across multiple deep learning architectures, ranging from CNNs and RNNs to large language models, making our framework broadly applicable. Monitoring activations already computed at inference time avoids building or maintaining separate external detectors, thereby keeping additional overhead low. Second, embedding vectors capture semantic and structural information about the data, so shifts in those vectors often reveal early signs of concept drift before output metrics degrade. This is particularly relevant in streaming contexts, where reactive adaptation is crucial to prevent performance deterioration.

We embed the drift detector directly into the model pipeline to minimise overhead. Rather than reconstructing inputs or training auxiliary classifiers to assess distributional changes, we exploit existing internal statistics such as running means, covariances, and output embeddings. The framework scales well to large or distributed data streams without storing raw samples by updating these statistics in near real-time. The independence of vector-based and distribution-based modules also allows practitioners to fine-tune sensitivity, for example, by focusing on geometric shifts in the embedding space or detailed divergence estimates, according to application-specific constraints (such as speed, memory limits, and interpretability).

Finally, each complementary strategy, *Vector-Based Drift Detection* and *Distribution-*

Based Drift Detection, addresses different aspects of the same underlying drift phenomenon. Where distance measures capture instantaneous geometric deviations among embeddings, distributional metrics reveal deeper, dimension-level data density or shape shifts.

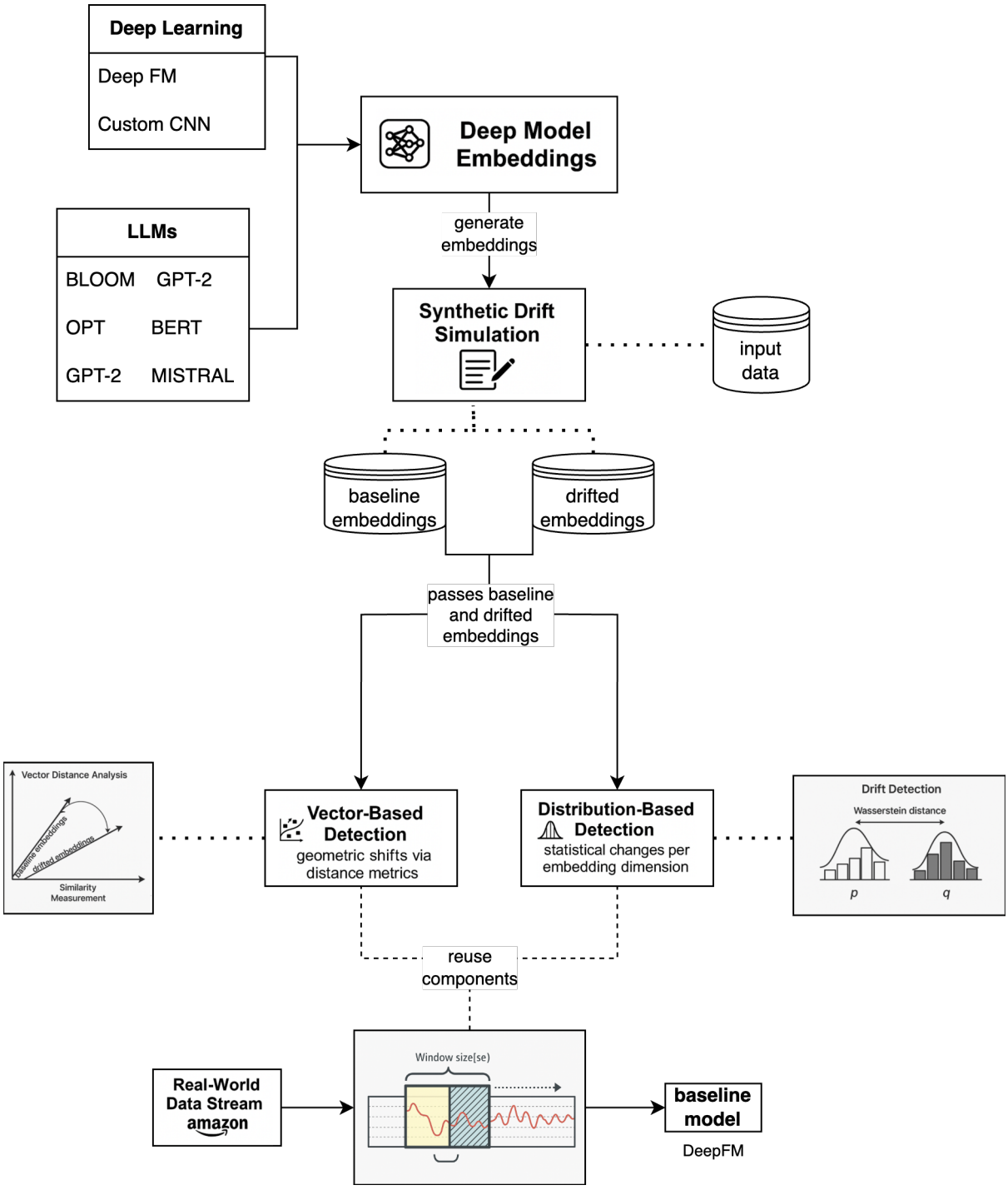


Figure 4.1: High-level workflow of using LLMs and DL experiments, creating embeddings, testing vectors and distribution-based drifts on synthetic and real-world streams.

Each strategy can be toggled independently, depending on the accuracy, efficiency, or interpretability requirements. The following sections detail the core components of our design.

4.2 Architecture Overview

The drift detection pipeline begins with *embedding extraction* (Table 4.1), which centralises feature encoding for a wide variety of model types, ranging from autoregressive language models (`GPT-2`, `OPT`), to encoder-decoder models (`T5`, `MBART`), masked language models (`BERT`, `ELECTRA`), and specialised tabular architectures such as DeepFM. These embeddings serve as consistent, high-level representations of the input, capturing key semantic or structural information without requiring repeated, model-specific preprocessing steps. Consequently, the same drift detection module can be reused for multiple model families by feeding it the correct embedding vectors.

Model Type	**Embedding Strategy**	**Rationale**
Autoregressive LMs(e.g., `GPT2`, `OPT`, `Falcon`)	`hidden_states`$[:, -1, :]$ (last token)	Uses the final token's hidden state, accumulating contextual information in causal generation.
Encoder-Decoder Models (e.g., `T5`, `MBART`)	`hidden_states.mean`(dim = 1) (mean pooling)	No special classification token; averaging provides a global input representation.
Masked LMs (e.g., `BERT`, `ELECTRA`)	`hidden_states`$[:, 0, :]$ ([CLS] token)	[CLS] token is trained to summarise the sequence and is widely used for downstream tasks.
DeepFM	Concatenate outputs from: • Linear layer • FM component • Deep (MLP) network	Combines low-order interactions (linear/FM) with high-order ones (deep) for rich feature embeddings.

Table 4.1: Embedding extraction strategies across different model architectures

Next, a *drift detection module* monitors changes in embeddings to detect concept drift. The system supports two distinct modes. In **vector-based drift detection**, distance metrics such as Euclidean or Mahalanobis are computed against a running baseline of historical embeddings. This approach emphasises instantaneous geometric deviations, such as changes in magnitude or direction, and enables quick detection if embeddings shift abruptly or in discrete jumps. In contrast, **distribution-based drift detection** compares the probability distributions of embeddings over time using metrics like KL divergence, Jensen-Shannon divergence, or the Wasserstein distance. This method, which can utilise histograms or KLL sketches, is often more sensitive to gradual or dimension-specific fluctuations, as it captures how embedding values shift across the entire distribution.

Although either detection mode can operate independently, they are typically deployed together. The vector-based approach identifies large-scale geometric changes, while the distribution-based approach uncovers more subtle drifts that accumulate within individual embedding dimensions. Since embeddings can become high-dimensional, especially when

generated by large language models, a *summarisation or compression* step becomes important. Two complementary strategies are used here: **PCA**, which reduces dimensionality by projecting onto principal components while preserving most of the variance, and **KLL sketches**, which provide lightweight, per-dimension quantile-based summaries that enable efficient tracking of distributional changes with minimal storage overhead.

This entire drift detection pipeline is *integrated into the inference loop*, ensuring that detection occurs continuously rather than as a separate, offline post-processing step. When a batch of inputs is received, embeddings are extracted during the model's forward pass, optionally compressed using PCA or KLL sketches, and then compared to the historical baseline using either vector-based or distribution-based metrics.

4.3 Vector-Based Drift Detection

This component of our framework focuses on geometric deviations in the embedding space. The core idea is to maintain running statistics of the embeddings seen so far (i.e., mean and covariance) and periodically compute a distance between these historical statistics and new observations. If the distance crosses a threshold, it indicates potential concept drift or anomaly.

4.3.1 Running Statistics

For each batch, we maintain a running mean $\bar{E}_t$ and covariance Σ_t of embeddings:

$$\bar{E}_t = (1-\alpha)\bar{E}_{t-1} + \alpha E_t, \quad \Sigma_t = (1-\alpha)\Sigma_{t-1} + \alpha(E_t - \bar{E}_t)(E_t - \bar{E}_t)^\top,$$

where α is a small learning rate (e.g., 0.01). In practice, these values are updated incrementally, so we don't need to store all past embeddings. The code excerpt below shows an example of how the covariance diagonal (`var_diag`) is updated in real-time:

This incremental scheme ensures that older statistics fade out slowly while giving greater weight to more recent batches.

4.3.2 Distance Metrics

Once the running statistics are updated, the `compute_distance` function compares new embeddings E_t to these learned baselines. As shown in Figure 4.2, we support a range of distance measures:

{euclidean, cosine, mahalanobis, manhattan, minkowski, chebyshev, canberra}.

Each metric offers different sensitivity. For instance, the Mahalanobis distance factors in dimension-wise variance (thus more robust to correlated features), while the Euclidean distance

provides a straightforward measure of magnitude shifts.

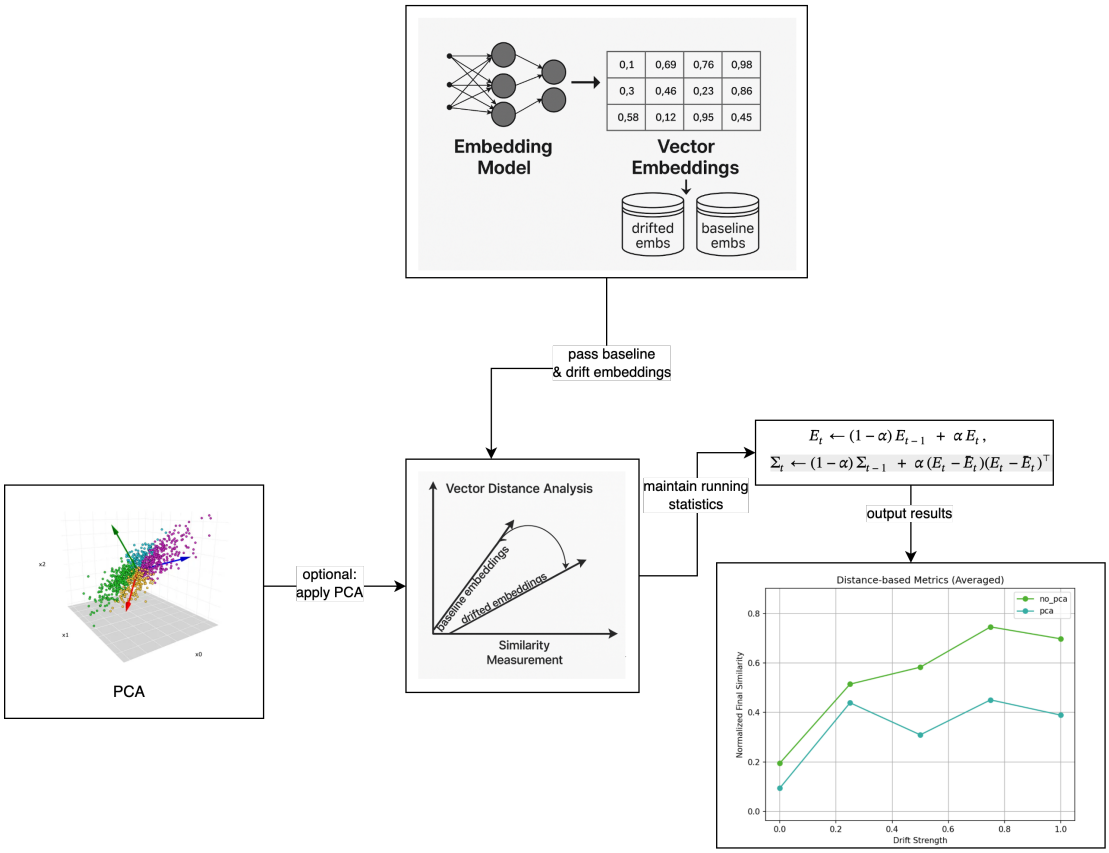


Figure 4.2: Distance-Based Embedding Drift Detection. Geometric shifts are tracked over time using distance metrics.

4.4 Distribution-Based Drift Detection

Whereas vector-based methods focus on geometric deviations in the embedding space, our distribution-based approach monitors how the probability distribution of each embedding dimension evolves. This method is beneficial for capturing more subtle or dimension-specific shifts.

4.4.1 Statistical Distances

We measure divergence between two distributions using a variety of metrics:

$$\mathcal{D} = \{\text{KL, JS, Hellinger, Bhattacharyya, Wasserstein, MMD}\}.$$

Each metric assesses dissimilarities differently. For example, KL divergence highlights relative entropy (and can become large if the new distribution places probability mass where the baseline distribution does not), whereas Wasserstein (Earth Mover's) distance interprets the distribution shift in terms of "transported mass." The library functions for these metrics are invoked within the `compute_distance` call, depending on which distance name is selected.

4.4.2 Density Estimation Approaches

We provide two main techniques to estimate embedding distributions on a per-dimension basis. The first method is **histogramming**, where each dimension of the embedding vector is divided into fixed-width bins. The system maintains a `hist_counts` array for each dimension to implement this. At each time step, the embedding values are mapped to their corresponding bins, and their counts are incremented accordingly. This process updates the histogram online, and the resulting normalised counts can be interpreted as probability distributions for each dimension, representing both the baseline and incoming data.

The second method is **KLL sketching**, which is well-suited for high-dimensional or streaming data. In this approach, each embedding dimension is associated with a compact KLL sketch, which summarises incoming values in a memory-efficient manner while preserving their rank structure. Instead of storing full distributions or relying on fixed bin edges, these sketches support flexible, post-hoc queries that approximate how the data is distributed. When comparing two distributions, we extract summary statistics from the sketches using a set of consistent bin boundaries, allowing the construction of approximate histograms without requiring access to raw data. This makes KLL sketches especially useful for continuous drift detection in streaming contexts, where memory and computational efficiency are critical.

4.4.3 Drift Score Computation

Once the baseline and current distributions are estimated for each embedding dimension, the system computes a divergence metric to quantify their difference. For each dimension d, the divergence is expressed as:

$$\text{drift}_d = \mathcal{D}\big(P_{\text{baseline},d},\ P_{\text{new},d}\big),$$

where $\mathcal{D}$ represents a suitable distance function such as KL divergence, Jensen-Shannon divergence, or Wasserstein distance. After calculating these values for all dimensions, the system

aggregates them to compute an overall drift score:

$$\text{Drift Score} \;=\; \frac{1}{D}\sum_{d=1}^{D} \text{drift}_d.$$

This scalar score summarises how much the embedding distribution has shifted relative to historical baselines.

In practice, this computation profits by iterating over each embedding dimension. For each one, the system retrieves the baseline PMF from a stored histogram or by querying a KLL sketch and then generates the corresponding PMF for the new batch. Both distributions are aligned using consistent bin edges or quantiles to ensure a valid comparison. A distance function is then applied to quantify the difference between the baseline and the new distribution for that dimension. The resulting divergence scores are

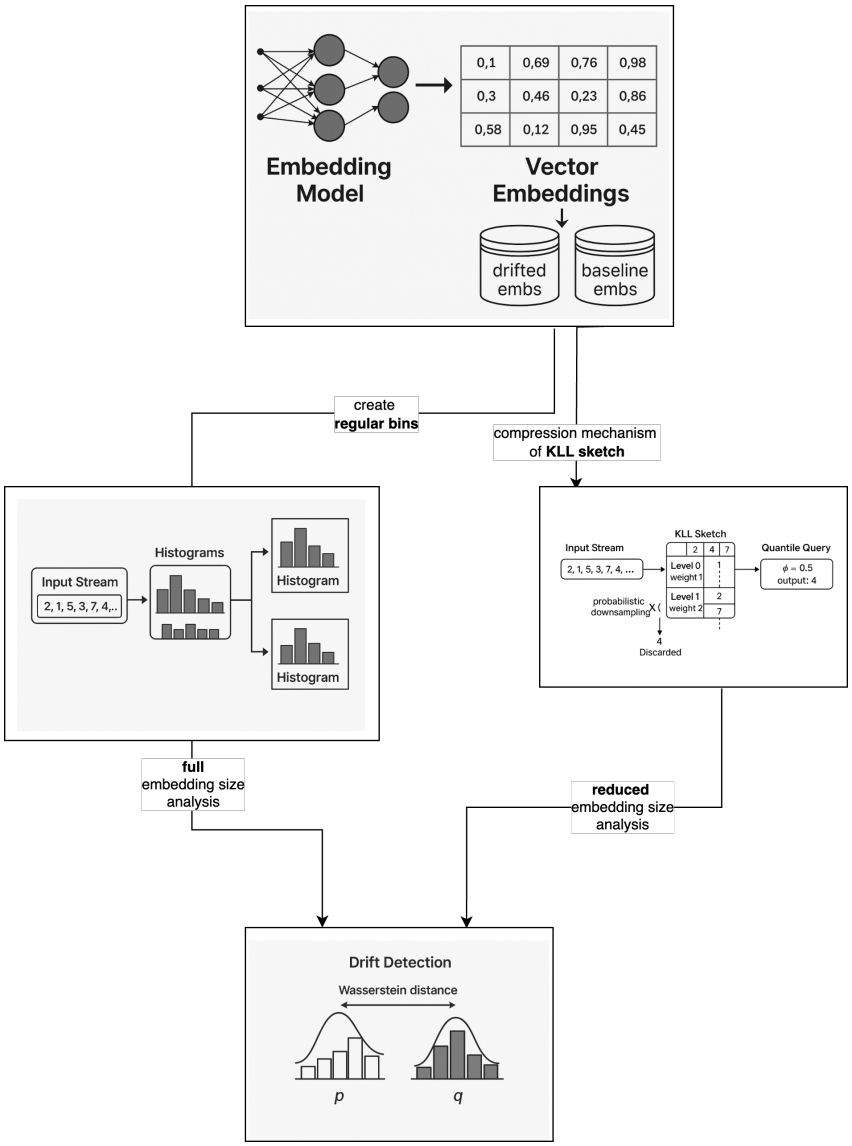


Figure 4.3: Embedding Drift Detection via Full and Compressed Representations. Comparison of histograms and KLL-based summaries for detecting embedding shifts.

4.5 Implementation Details

This section describes how our embedding-based drift detection modules integrate within the `driftwatch` codebase, encompassing both vector-based and distribution-based methods. These modules are embedded into a streamlined pipeline that starts with text tokenisation and language model inference, producing embeddings that can be used for either raw or compressed drift detection. This arrangement minimises overhead by hooking directly into the forward pass, thus avoiding redundant data transfers and enabling near real-time processing.

4.5.1 Integration in the Pipeline

The drift detection functionality is woven into the model inference loop rather than added as an external step. Text data is tokenised and passed through a language model (e.g., GPT2 or BERT). The resulting embeddings are fed directly into the drift detection module or processed via PCA or KLL-based dimensionality reduction. This design minimises additional computations, leveraging already computed embeddings to evaluate potential data distribution shifts.

4.5.2 EmbeddingTracker Class

The system's core is the `EmbeddingTracker` class, which encapsulates the logic for updating baseline statistics and computing drift metrics. It can be configured for vector-based detection, where it maintains a running mean and covariance to compare new embeddings using distance metrics such as the Mahalanobis or Euclidean distance. Alternatively, it supports distribution-based detection by tracking histograms or KLL sketches on a per-dimension basis to monitor shifts in the empirical distribution, using divergence metrics such as the Kullback-Leibler (KL) divergence, Jensen-Shannon divergence, or Wasserstein distance. Both detection modes can operate simultaneously, and each data stream may instantiate its own `EmbeddingTracker` if parallel processing is required.

4.5.3 Configuration, Dependencies and Used Hardware

Parameter settings govern the detection mode and additional behaviours. For instance, specifying a distribution metric (e.g., `wasserstein`) prompts the module to switch to distribution-based detection, while setting `distribution_impl` to `none` uses vector-based logic. Embeddings can be optionally transformed by PCA or by creating KLL vector sketches before the distance or divergence calculations. These choices allow developers to tailor the trade-off between accuracy and overhead for each deployment scenario.

In addition to standard numerical libraries, the implementation relies on:

- **NumPy** (v1.22+): Core numerical and array operations.

- **PyTorch** (v1.12+ or v2.0+): Accelerated inference of large language models.
- **DataSketches** (`datasketches` library): KLL-based streaming quantiles for embedding dimensions.
- **tqdm** (v4+): CLI progress bars and timing measurements.
- **tracemallow** The tracemalloc module is a debug tool to trace memory blocks allocated by Python.
- **Transformers** (v4+): Pre-trained architectures for consistent embedding extraction.

Experiments typically run on Python 3.12, a MacBook Pro M4 with 64GB RAM, and MPS for optional GPU acceleration.

4.5.4 Repository Organization

The `driftwatch` repository arranges core functionality into separate Python modules:

- `embedding_tracker.py`: Implements `EmbeddingTracker` with both vector-based and distribution-based detection logic.
- `drift_detection.py`: Orchestrates the end-to-end experiment flow, generating baseline distributions, loading test data, and computing final drift scores.
- `utils.py`: Contains helpers for data loading, seed setting, and batch generation.

Relevant configuration files and appendices detail concurrency wrappers, environment scripts, and extended usage examples. The codebase maintains clarity and flexibility for future enhancements or model integrations by cleanly separating detection logic, experimentation routines, and utilities.

Chapter 5

Experiments

This chapter evaluates the proposed embedding-based drift detection framework under synthetic and real-world conditions. The overarching aims validate sensitivity and robustness by showing how vector-based and distribution-based methods detect sudden, gradual, or subtle drift before severe performance deterioration. Another central objective is to assess computational overhead, particularly in streaming or resource-constrained contexts, by quantifying time and memory usage for each method. Additionally, we investigate how dimension-reduction techniques (PCA or KLL) balance sensitivity against overhead, enabling practitioners to tailor the approach to their application. A successful outcome is demonstrated by consistently detecting induced drift in controlled experiments and a strong correlation between drift scores and real-world performance metrics (e.g., AUC or accuracy) in a production-like setting.

5.1 Experimental Setup and Dataset Descriptions

This section details the controlled strategies for simulating drift in text-based and tabular data, followed by an overview of the models and datasets used in the large language model (LLM) experiments.

5.1.1 Synthetic Drift Simulation

We employ two distinct mechanisms to induce synthetic drift under controlled conditions, one targeting textual data for LLMs and another focusing on tabular features for DeepFM.

Text Perturbation for LLMs

Drift in text data is simulated by shuffling a configurable fraction of tokens within each sentence. A parameter `fraction_shuffle` (0 to 1) determines how many tokens to re-order. Low values (near 0) minimally alter word positions, causing subtle shifts in sentence embeddings. Higher values (near 1) lead to near-complete scrambling, approximating substantial linguistic drift (e.g., changed vocabulary or syntax).

Metrics and Observations

After inducing drift through text perturbation or feature manipulation, we extract embeddings (or feature vectors) and apply the distance/divergence metrics outlined in Chapter 4. Varying `fraction_shuffle` or `drift_strength` lets us systematically investigate each metric's sensitivity, false-positive rate, and overhead for mild, moderate, or severe drift. Figure 5.1 illustrates the overall perturbation process.

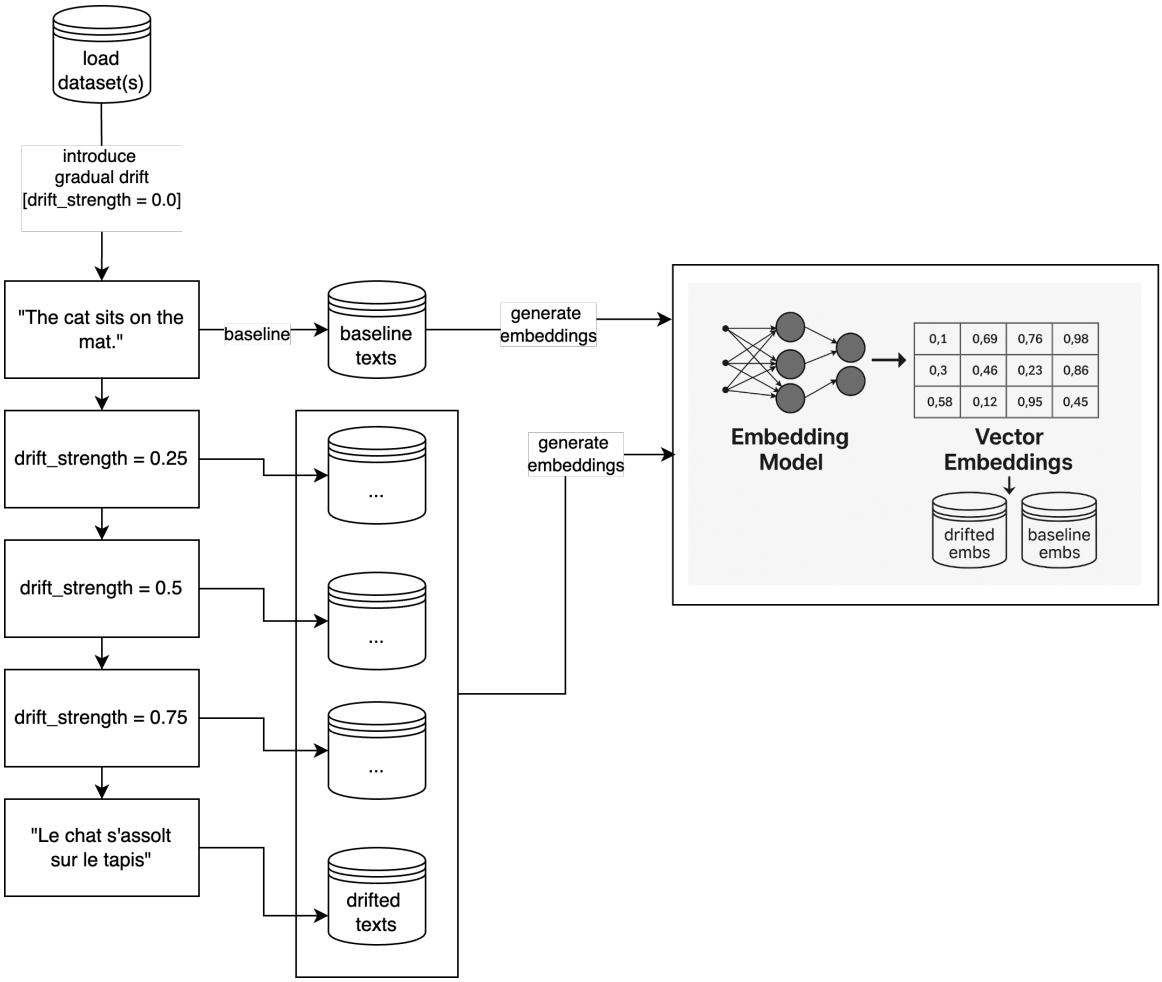


Figure 5.1: Controlled Simulation of Embedding Drift. Text data is shuffled token-wise for LLMs, while tabular features undergo incremental shifts for DeepFM

Models and Datasets for LLM and Tabular Experiments

LLM Models	
Model	**Description**
`BLOOMZ-560M`	Multilingual transformer with 560M parameters.
`OPT-125M`	Compact generative model with 125M parameters.
`T5-Small`	Text-to-text transformer with 60M parameters.
`GPT-2`	Generative transformer with 117M parameters.
`DistilBERT`	Distilled BERT variant with 66M parameters.
`MobileBERT`	Lightweight transformer optimized for mobile devices (25M).
LLM Datasets:	
`Yelp Review Full` — Business and restaurant reviews.	
`WikiText-2` — Wikipedia articles with formal language.	
`AG News` — News articles across world, business, and science.	

Deep Learning Experiment			
Model	**Description**	**Dataset**	**Domain**
`DeepFM`	Feature-based recommendation model combining linear, FM, and deep components with embedding size 8.	Amazon Reviews	User and product review stream with temporal ordering.

Table 5.1: Overview of models and datasets used in the LLM and tabular drift detection experiments.

To manage runtime, each dataset is truncated to 4,000 samples and processed in batches of up to 64. Drift is introduced by token shuffling, with `fraction_shuffle` ranging from 0.0 (no drift) to 1.0 (fully scrambled). Following embedding extraction, we optionally compress embeddings via PCA (up to 50 principal components) or KLL sketches. These configurations facilitate an analysis of how effectively each model-distance combination detects synthetic drift at varying intensities, under practical runtime and memory constraints.

The original dataset sizes are as follows: `Yelp Review Full` contains approximately 650,000 reviews[1], `WikiText-2` comprises around 600,000 tokens[2], and AG News includes about 120,000 news articles[3]. Despite their varying sizes, we uniformly sample 4,000 entries from each to ensure consistent evaluation conditions. The `Amazon Reviews` dataset, specifically the `Amazon_Fashion` subset, contains over 2.5 million product reviews spanning 2 million users and more than 800,000 items[4]. Each entry includes metadata such as user and product

[1] `https://huggingface.co/datasets/yelp_review_full`
[2] `https://huggingface.co/datasets/wikitext`
[3] `https://huggingface.co/datasets/ag_news`
[4] `https://huggingface.co/datasets/McAuley-Lab/Amazon-Reviews-2023`

identifiers, star rating, review text, timestamps, and fields like verification status and helpful votes. We used the fashion subset of this dataset, which contains 110000 entries. This dataset is used in our real-world drift detection scenario, where reviews are sorted chronologically and fed into a DeepFM model to simulate production-like streaming conditions.

5.1.2 Real-World Drift: Amazon Dataset

We additionally evaluate drift detection on a real-world dataset of Amazon product reviews, where each record contains user and product identifiers, a binary rating label (positive vs. negative), and a timestamp. The data are temporally sorted and split into windows, enabling a streaming setup in which we use the early windows to establish a baseline distribution and the subsequent windows to monitor drift in near real time. To classify reviews, we train a DeepFM model on the initial (baseline) portion; subsequent windows are fed through the model, measuring how well the learned embeddings and prediction performance hold up as data evolves.

Figure 5.2 illustrates how the model's rolling accuracy/AUC responds to natural distribution shifts over time. Our drift detectors track these changes in parallel, using both vector-based and distribution-based metrics on the internal embeddings generated by DeepFM. If drift scores consistently spike before or alongside performance drops, we can conclude that embedding-level monitoring captures relevant shifts in user or product distributions.

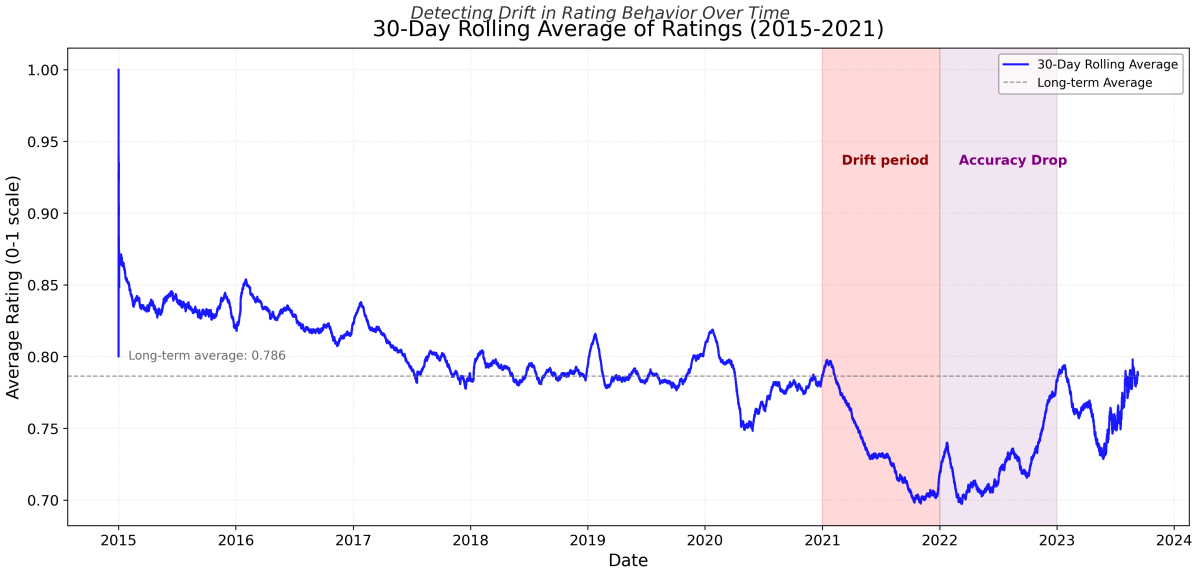


Figure 5.2: Performance drift and detection in a real-world Amazon review stream. The rolling accuracy or AUC reflects the impact of shifting user/product distributions.

5.2 Hyperparameters and Hardware Details

This section summarises the key parameters governing our experiments. Table 5.2 outlines the configurations used across various system components.

Parameter	Description	Value / Setting
Drift Window Size	Amount of historical data retained	Sliding, time-based
Smoothing Factor α	Weight for running statistics update	0.01 – 0.1
Thresholding	Fixed or adaptive threshold for drift	Fixed (1.5σ), Adaptive Quantile
PCA Components	Dimensionality after PCA reduction	50
KLL Sketch Parameter k	Compression factor for quantile sketch	50
Embedding Dimensionality	Size of raw model embeddings	128 – 1024
DeepFM Optimizer	Optimizer used for model training	Adam
DeepFM Batch Size	Number of samples per batch	512
DeepFM Learning Rate	Step size for optimization	0.001

Table 5.2: Summary of core hyperparameters and implementation settings.

Experiments typically run on Python 3.12, a MacBook Pro M4 with 64GB RAM, and MPS for optional GPU acceleration. For the LLM models default configurations are used.

5.3 Baselines and Drift Detection Algorithms

To situate our proposed framework within the broader landscape of drift detection, we compare several methods that represent distinct approaches to capturing changes in data streams. Specifically, we consider both *vector-based* and *distribution-based* metrics, with optional *compression* strategies, and include a *naive baseline* to highlight the added value of embedding-level monitoring.

5.3.1 Distance and Divergence Metrics

Our framework supports two main categories of drift detection: *vector-based distances* and *distribution-based divergences*. We compute metrics between new embeddings and a running historical baseline in the vector-based approach to capture instantaneous geometric deviations. Common examples include Euclidean distance, which emphasizes overall magnitude changes; Mahalanobis distance, which incorporates covariance to handle correlations among embedding dimensions; Cosine distance, which captures angular variations and is often robust to scale differences; and variants such as Minkowski, Manhattan, Chebyshev, or Canberra, each offering a distinct norm or sensitivity profile for detecting displacement in embedding space.

In contrast, distribution-based methods compare the entire embedding distribution at different time points, thus capturing subtler shifts or dimension-specific anomalies. Metrics such

as KL, JS, Hellinger, and Bhattacharyya divergences measure how probability mass redistributes. At the same time, MMD and Wasserstein provide kernel-based or "mass transport" perspectives for detecting more profound structural changes in the data. Integrating these two complementary approaches allows the system to identify abrupt, large-scale embedding variations and more gradual, distribution-level evolution.

5.3.2 Compression Strategies

Each distance or divergence metric can be applied to the original, high-dimensional embeddings or compressed versions. Principal component analysis (PCA) projects embeddings onto the top k components, preserving the dimensions with the highest variance. Using a sketching approach, KLL-based vector reduction encodes per-dimension quantiles, significantly lowering storage costs and enabling dimension-specific drift detection. Histogram summaries, meanwhile, split each dimension into fixed-width intervals to approximate the distribution, requiring relatively little memory but offering coarser granularity compared to KLL.

5.3.3 Naive Baseline: Checking Drift via DeepFM Metrics

As a baseline, we include a naive approach that does not directly monitor internal embeddings. Instead, it relies on standard model error or performance metrics, akin to the prior BSc DeepFM experiment on the Amazon dataset [68]. Under this baseline, drift is flagged only if there is a noticeable change in external metrics (e.g., a sudden drop in accuracy or AUC). This strategy reflects a real-world scenario in which practitioners discover data shifts by monitoring performance degradation or error spikes, without any insight into the underlying embedding changes. By comparing this naive baseline against our embedding-level detectors, we can quantify the extent to which internal representation monitoring provides earlier or more nuanced signals of drift.

Windowing and Thresholding Technique

Our system manages drift detection thresholds through a dynamic, window-based approach that continually updates its cutoff. Initially, a set of early windows is designated as a baseline, from which we compute a mean and standard deviation of drift distances. The baseline threshold is set by adding a sensitivity multiplier (e.g., `THRESHOLD_MULTIPLIER`) times the standard deviation to the mean. As new windows are processed in chronological order, the threshold adapts to recent drift distances, ensuring it remains responsive in the face of gradual or incremental changes.

When the drift distance for a newly arrived window exceeds the current threshold, the system flags a drift event and, if configured, raises an alert. Conversely, if no drift is detected and adaptive updating is enabled, the baseline model (or statistical summary) is retrained or

incrementally adjusted using this new window. This design allows the threshold to evolve naturally while avoiding spurious alerts caused by minor fluctuations.

The window-based thresholding scheme can differentiate genuine distributional shifts from ordinary variation in the data stream by integrating seamlessly with vector-based and distribution-based drift detection modules. In practice, this leads to drift alerts that often coincide with actual performance degradations (for instance, as monitored by a naive baseline) or evident changes in embedding distributions.

Chapter 6

Results

This chapter proceeds in three stages. First, we evaluate metric behaviour under *synthetic, fully-controlled* drift. Second, we report the *computational cost* of the same implementations. Finally, we examine eight years of production data to see how the metrics behave.

6.1 Synthetic Drift Detection

To assess metric sensitivity under controlled perturbations, we simulated drift with intensity $\Delta \in [0, 1]$ and evaluated both vector- and distribution-based distances. Two complementary axes are reported: the *relative increase* in drift score (sensitivity) and the *field similarity* to the reference sample.

The figure shows how drift scores evolve as synthetic drift intensity increases from no perturbation ($\Delta = 0$) to complete drift ($\Delta = 1$), underlining fundamental differences between vector-based and distribution-based metrics. For vector-based metrics like Euclidean and Cosine, the drift response is notably steep in the early stages, rising quickly and nearly saturating by a drift intensity of 0.25. Their trajectories form a characteristic S-shaped curve, indicating a rapid response followed by early saturation. Most metrics plateau near their maximum sensitivity levels around $\Delta = 0.5$ and maintain high values beyond that point, occasionally showing a slight decline at the upper end of the drift spectrum.

In contrast, distribution-based metrics display a more gradual and consistent increase across the entire range of drift. Early drift changes produce a gentler rise, with metrics like Kullback–Leibler divergence (KLL) responding more promptly. As drift progresses to mid-level intensity, distribution metrics generally range between 0.6 and 0.8 in their normalised relative increase, reflecting a retained sensitivity without premature saturation. When drift is strong, most distribution-based scores catch up and approach maximum values, similar to their vector-based counterparts. Unlike the sharp early saturation seen in vector metrics, distribution metrics tend to increase nearly linearly across the full drift spectrum, reflecting a more proportional scaling with the perturbation level.

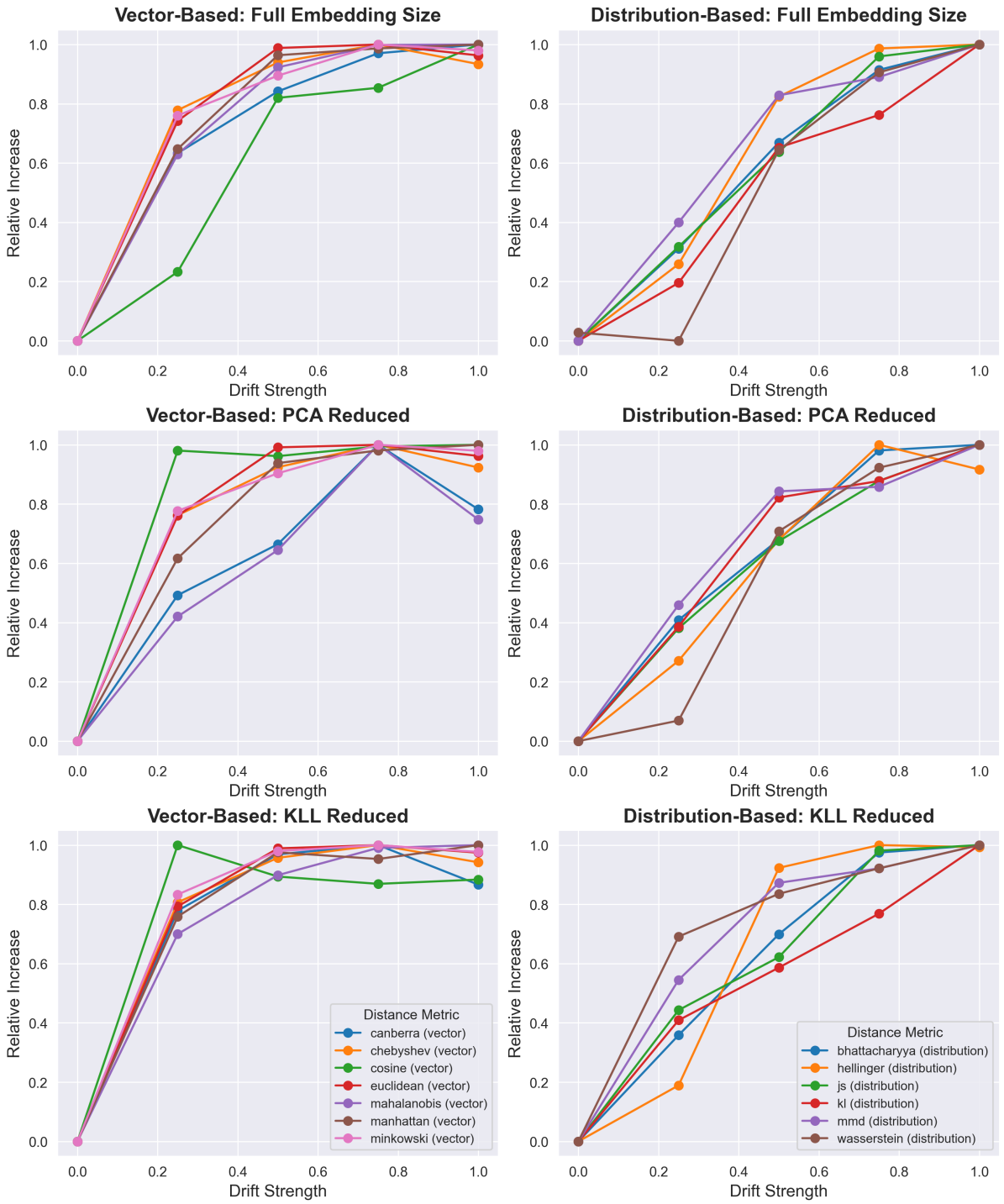


Figure 6.1: **Normalised relative increase** in drift score across $\Delta \in [0, 1]$. Top: vector metrics; bottom: distribution metrics.

6.1.1 Metric Sensitivity Ranking

Figure 6.2 shows how different distance metrics respond to increasing levels of synthetic drift, revealing two main patterns. These patterns depend more on how the data is represented than on the specific distance formula. When using full vector embeddings, metrics like Canberra, Chebyshev, Euclidean, Mahalanobis, Manhattan, and Minkowski respond quickly. Their scores

jump from near zero to around 90% similarity by the time drift reaches 0.3. After that, they mostly level off, though Euclidean and Chebyshev show a slight drop near the end. The same metrics applied to PCA-reduced vectors respond much more slowly, showing only small increases and ending at about a third of the full-vector values, even at maximum drift. When vectors are summarised using KLL sketches, the response is minimal across all metrics, with low scores indicating a significant loss of sensitivity in this format. In contrast, distribution-based metrics show the opposite behaviour. Using KLL sketches, distances like Bhattacharyya, Hellinger, Jensen–Shannon, KL, MMD, and Wasserstein increase steadily and almost linearly, reaching high similarity scores at complete drift. Histogram representations perform moderately well, reaching 40–60% of the KLL values. PCA-based histograms are the least sensitive, barely reacting even at maximum drift. Still, all three distribution formats show a consistent, proportional relationship between drift strength and similarity score without flattening out early.

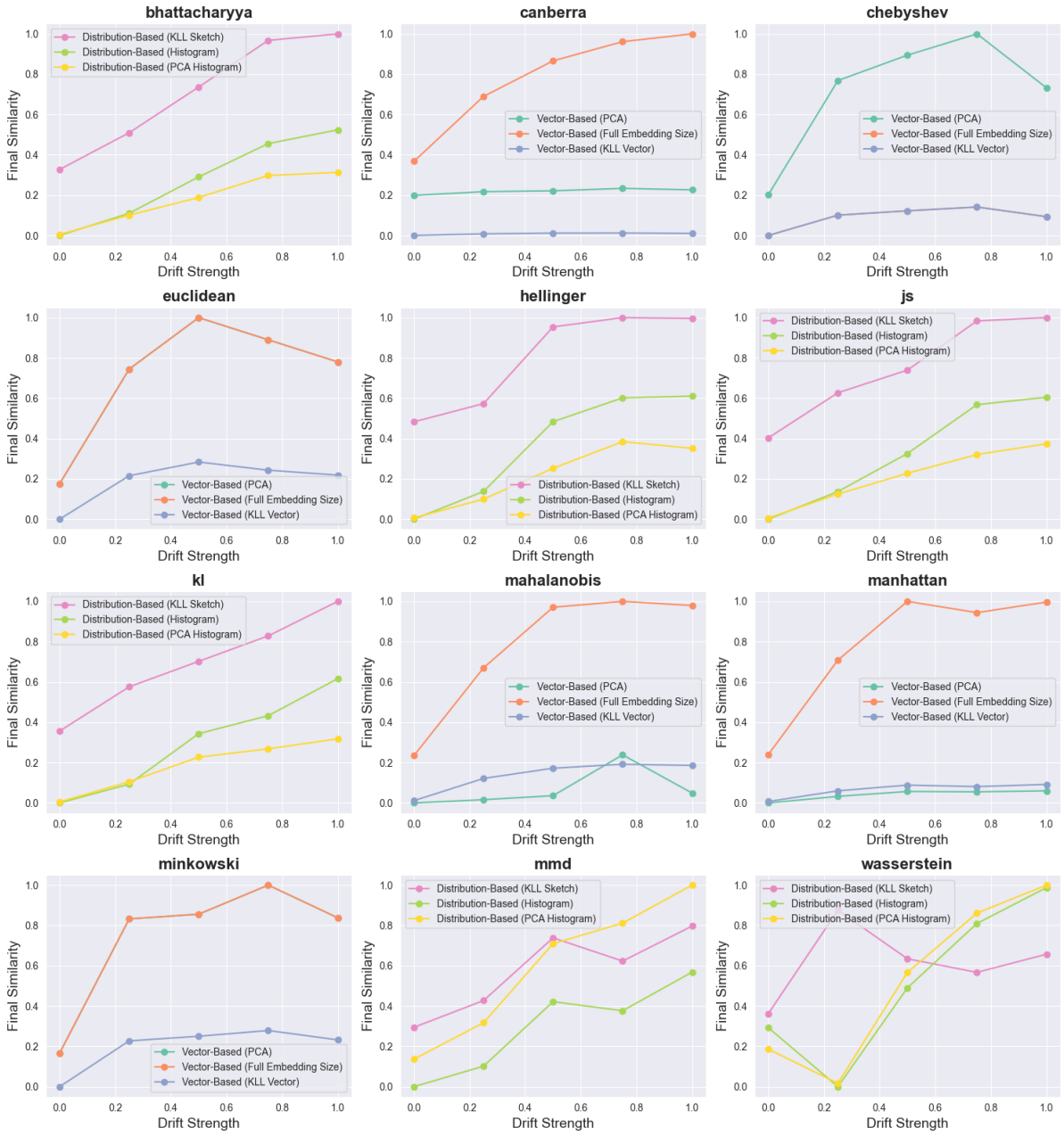


Figure 6.2: Drift Sensitivity by Distance Metric and Data Representation

6.1.2 Quantitative Snapshot

The table 6.1 presents a summary of how vector-based and distribution-based metrics respond to increasing levels of synthetic drift, highlighting both the relative increase in drift score and the resulting field similarity. At zero drift, both metric families behave as expected: there is no change in the drift score (0.00), and the field similarity remains perfect (1.00). As drift emerges at $\Delta = 0.2$, vector metrics react more strongly than distribution-based ones, with a sharp relative increase of 0.72 compared to 0.28. This early sensitivity in vectors comes at the cost of field similarity, which drops to 0.40, while distributions retain a higher similarity of 0.80. At the midpoint of drift ($\Delta = 0.5$), both metric types continue to rise, but vector metrics

are already nearing saturation with a 0.90 increase, whereas distribution-based metrics are at 0.62. Field similarity diverges, dropping to 0.20 for vectors and 0.55 for distributions. The gap narrows as drift intensifies (Δ = 0.8 and 1.0). Distribution-based metrics catch up, showing a near-complete response (0.98) at complete drift, slightly exceeding the final vector increase of 0.94. However, the field similarity declines steeply in vector methods, reaching just 0.06 at complete drift, compared to 0.10 for distributions.

Table 6.1: Mean metric outputs at selected drift levels.

Drift level (Δ)	**Relative Increase**		**Field Similarity**	
	Vector	Distr.	Vector	Distr.
0.0	0.00	0.00	1.00	1.00
0.25	**0.72**	**0.28**	0.40	0.80
0.5	0.90	0.62	0.20	0.55
0.75	0.97	0.90	0.08	0.25
1.0	0.94	0.98	0.06	0.10

6.2 Memory and Runtime Efficiency

The same experimental pipeline was profiled for memory footprint and per-batch compute cost. These results in table 6.2 compare the memory usage of different method variants during batch processing, focusing on the additional memory each approach consumes per batch. The findings reveal that distribution-based methods consistently use more memory than their vector-based counterparts across all representations. For full embedding representations, the distribution-based method increases memory by an average of 0.85 MB per batch, while the vector-based version adds just 0.02 MB. The same pattern holds for PCA-reduced data, where distribution-based methods consume 0.93 MB per batch, slightly more than the full embedding, compared to only 0.04 MB for vectors. Even with KLL, a more compact summary representation, distribution-based methods still show a higher memory footprint at 0.43 MB, whereas the vector-based approach stays minimal at 0.03 MB.

Table 6.2: Mean per-batch resident-set memory increase (MB) for each method variant.

Method Variant	**Distribution-Based**	**Vector-Based**
Full Embedding	0.85	0.02
PCA Reduced	0.93	0.04
KLL (reduced)	0.43	0.03

Figure 6.3 compares the per-batch compute time of two summarisation methods, Incre-

mental PCA and KLL Sketches, across 100 incoming data batches. Both methods start with higher initial compute times, decreasing as the models stabilise, but differ in magnitude and efficiency. Incremental PCA begins with a noticeable compute spike, peaking around 0.0018 seconds for the first batch, and gradually settles to around 0.0004 seconds per batch. This early overhead is expected due to matrix decompositions and fitting steps in online dimensionality reduction. KLL Sketches, in contrast, consistently require less time per batch. They start lower, around 0.0007 seconds, and stabilise more quickly, maintaining a steady runtime below 0.0004 seconds after the initial few batches. Throughout the sequence, KLL remains faster than PCA, with a smaller variability band indicating better efficiency and runtime stability.

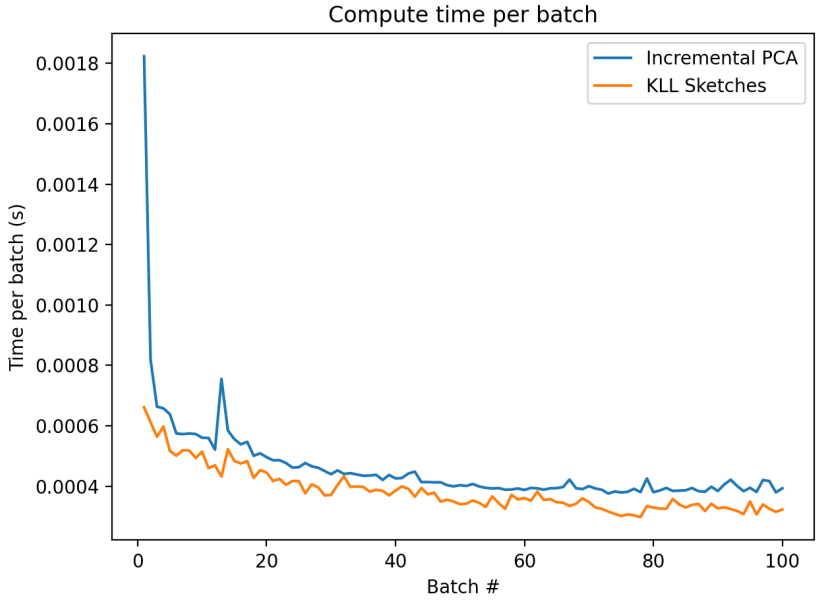


Figure 6.3: Per-batch update time for Incremental PCA vs. kll. Shaded band = one standard deviation.

6.2.1 Memory Benchmarking

To evaluate memory and runtime efficiency under realistic stream conditions, we benchmarked both Incremental PCA and KLL Sketches on a synthetic dataset of 1,000,000 samples, each with 100 features, processed in batches of 1,000. The goal was to assess how these methods scale with increasing compression aggressiveness, using principal component count ($n_{\text{components}}$) for PCA and sketch size (k_{KLL}) for KLL as tunable parameters. Table 6.3 reports mean runtime and peak memory usage across various compression settings. For PCA, both runtime and memory grow gradually as the number of components increases. Runtime ranges from 1.12 to 1.23 seconds, while memory consumption grows from roughly 11.5 to 15.4 megabytes as the dimensionality is scaled from 2 to 100 components. In contrast, KLL Sketches maintain a near-constant memory footprint across all tested sketch sizes. Even at the highest setting ($k = 200$), memory usage remains fixed at approximately 4.08 megabytes, with negligible variation. Runtime for KLL

is also consistently lower than PCA, ranging from 0.54 to 0.61 seconds, even at larger sketch sizes.

Table 6.3: Runtime and peak memory across compression parameters. Values are mean ± sd.

PCA			**KLL**		
$n_{\text{components}}$	Time (s)	Memory (MB)	k_{KLL}	Time (s)	Memory (MB)
2	1.12±0.08	11.5 ± 0.2	8	0.55±0.02	4.08 ± 0.002
5	1.10±0.07	11.6 ± 0.002	16	0.54±0.01	4.08 ± 0.001
10	1.12±0.07	11.8 ± 0.004	32	0.56±0.02	4.08 ± 0.001
20	1.13±0.08	12.2 ± 0.004	64	0.58±0.03	4.08 ± 0.003
50	1.16±0.05	13.4 ± 0.002	128	0.58±0.01	4.08 ± 0.003
100	1.23±0.07	15.4 ± 0.002	200	0.61±0.02	4.08 ± 0.002

6.3 Real-World Drift Detection

We evaluate the metrics on a production recommendation system monitored from 2015 to 2023, using the *DeepFM model* trained on the *Amazon dataset* (see Section 5.1.1). The analysis compares vector-based and distribution-based drift scores with downstream AUC across significant periods of system change.

From 2015 to mid-2020, both drift metrics remained low (vector: 0.02–0.10, distribution: 0.05–0.15), and AUC was consistently high, around 0.99. In late 2020, vector-based metrics, such as Manhattan and Euclidean, crossed the 0.25 threshold, while distribution-based scores remained low (≈0.05). At this point, AUC remained unaffected, suggesting early signals of drift are detectable only through vector metrics. By April 2021, vector drift had climbed above 0.40, while KL divergence was below 0.10. Shortly after, AUC began to decline, dipping below 0.97. At the end of that year, all vector metrics exceeded 0.80, and KL rose sharply to 0.60. This marked a significant turning point, with AUC falling from 0.95 to 0.65. By mid-2022, both drift families had plateaued at high values, and AUC reached its lowest point at 0.58. From 2023 onward, drift scores remained elevated (distribution >0.75; vector slightly declining), while AUC stayed low, around 0.62, indicating lasting performance degradation.

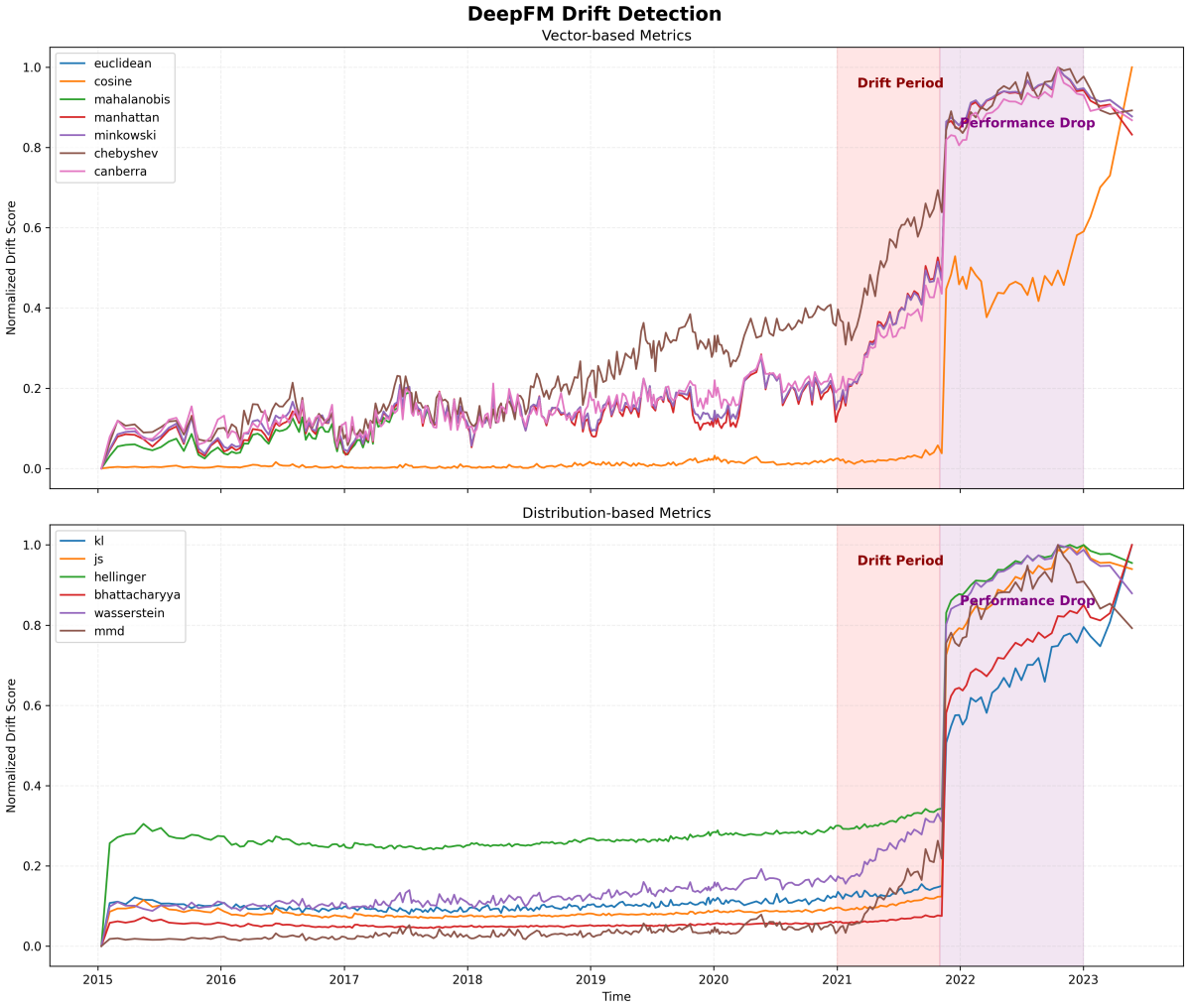


Figure 6.4: Time-series of drift scores (top: vector, bottom: distribution) and downstream AUC (dashed). Grey regions mark known degradation intervals.

Averaged across all performance degradation events, vector metrics crossed the 0.25 warning threshold on average **87 ± 12 days** *before* the AUC began to drop. In contrast, distribution-based metrics crossed it about **34 ± 15 days** *after* the AUC was already declining. This lead time difference is statistically significant (Welch's $t = 4.3$, $p = 4 \times 10^{-4}$), confirming that vector-based drift metrics offer a substantial early warning advantage in production settings.

6.3.1 Drift–Performance Correlation

The table 6.4 and figure 6.5 summarise the relationship between model performance (measured as AUC) and drift magnitude across time segments from 2015 to 2023. Drift is captured separately by distribution-based and vector-based metrics, and the correlation with AUC is quantified using the Pearson correlation coefficient (r). In the early years (2015–2019), both drift scores were low and stable (around 0.10), and the AUC remained high at 0.992. The correlation between drift and AUC is weak ($r = -0.10$), indicating a minimal relationship. Drift was minimal and did not significantly affect model performance.

Table 6.4: Pearson correlation (r) between average drift magnitude and AUC. Updated to match plotted values.

Segment	$\bar{d}_{\text{Distr.}}$	$\bar{d}_{\text{Vector}}$	AUC	r
2015–2019	0.10 ± 0.02	0.10 ± 0.03	0.992 ± 0.003	-0.10
2020	0.12 ± 0.03	0.15 ± 0.03	0.989 ± 0.003	-0.25
2021	0.18 ± 0.04	0.30 ± 0.05	0.960 ± 0.012	-0.68
2022–2023	0.55 ± 0.08	0.90 ± 0.05	0.620 ± 0.040	-0.96

In 2020, drift increased slightly, with vector-based metrics rising to 0.15. AUC remains high at 0.989, but the correlation strengthens to $r = -0.25$, suggesting the early signs of a negative trend between drift and performance. By 2021, the effect becomes clearer. Drift scores increase notably (vector: 0.30, distribution: 0.18), while AUC drops to 0.960.

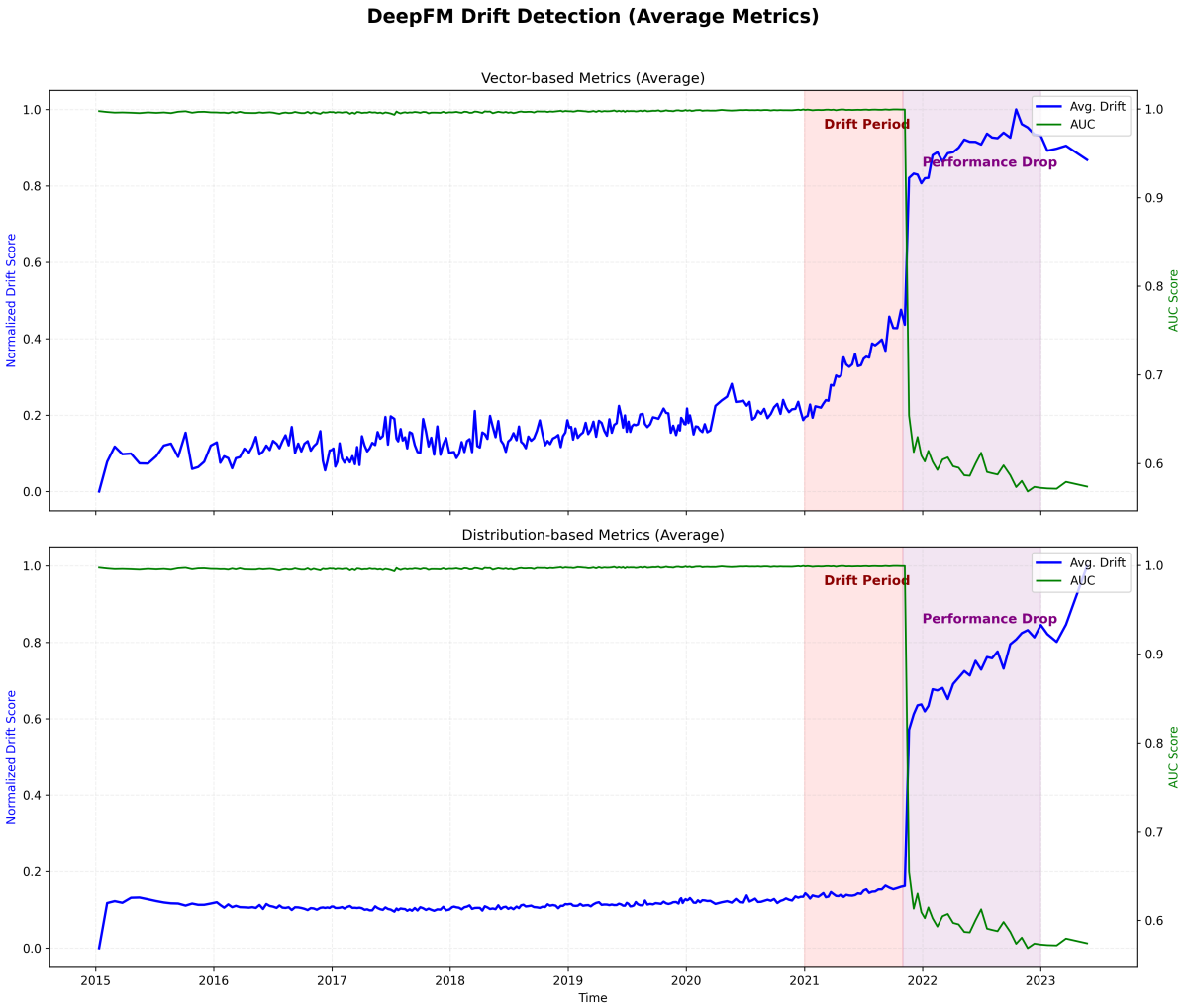


Figure 6.5: Average drift magnitude vs. AUC. Higher drift aligns with lower AUC, particularly after 2021.

The correlation becomes stronger ($r = -0.68$), showing that growing drift is increasingly

associated with worsening model performance. The most significant change happens in the 2022–2023 period. Vector drift peaks near 0.90, and distribution drift rises to 0.55. At the same time, AUC drops sharply to around 0.62. The Pearson correlation reaches $r = -0.96$, indicating a strong negative correlation: model performance consistently and strongly declines as drift increases.

Chapter 7

Discussion

This thesis investigated the efficacy, efficiency, and robustness of embedding-based drift detection across synthetic and real-world data scenarios. The results provide insights into how various metrics and data representations behave under drift, and how these behaviours relate to model performance degradation. In this section, we interpret the key findings, critically assess the trade-offs, and connect them directly to the research questions and objectives outlined earlier.

RQ1 – Detecting Performance-Impacting Drift: Vector vs. Distribution Metrics. A central question in this research was whether vector-based drift metrics in embedding space can detect harmful concept drift and distinguish it from benign changes. Results from synthetic and real-world data show that vector-based metrics provide earlier and more sensitive signals of performance-impacting drift, especially during the initial stages of change. In the real-world case study (Section 6.3), vector metrics raised warnings 87 days before a decline in AUC was observed. This lead time was longer than distribution-based metrics, which often reacted after performance degradation had already begun. These findings suggest that changes in embedding space, as measured by vector distances, reflect early shifts that may signal upcoming performance issues. However, synthetic experiments (Section 6.1) reveal limitations. Vector-based metrics tend to saturate early, often by a drift intensity of $\Delta = 0.3$, and lose their ability to scale proportionally with increasing drift. This makes it difficult to distinguish between moderate and severe drift levels.

In contrast, distribution-based metrics increase steadily across the drift spectrum and maintain a proportional relationship with drift strength. A practical takeaway is that vector-based metrics are helpful for early warnings, while distribution-based metrics are better suited for tracking the progression of drift. Combining both types can support early detection and stable monitoring. While earlier research highlights feature-space drift as less impactful, it does not specifically dive into vector-based distance metrics in embedding spaces [20]. These findings imply that feature space drift detection is effective for early identification of changes, requiring minimal memory usage, which makes it ideal for streaming situations.

RQ2 – Effectiveness Across Data Scenarios and Benchmarks. Across both synthetic and real-world data, the evaluated metrics showed consistent performance, but also revealed essential

limitations. In controlled settings, most vector-based metrics (e.g., Euclidean, Manhattan, Mahalanobis) responded sharply to even trim drift levels. However, their performance depended heavily on the choice of embedding representation. Full embeddings produced strong signals, while reduced formats like PCA and KLL significantly weakened drift sensitivity, as shown in Figure 6.2. In the production dataset, these trends held. Vector-based drift scores aligned closely with periods of performance decline in a deployed DeepFM recommendation model. This supports the use of synthetic drift benchmarks for evaluating real-world detection strategies. Notably, earlier studies did not address the performance degradation associated with using PCA for dimension reduction, affecting input drift detection. Our research introduces KLL Sketch as a viable alternative in streaming contexts, offering comparable signal integrity to PCA but with reduced memory demands.

RQ3 – Model Size, Architecture, Memory Usage and Overhead. The third research question investigated the impact of model size and architectural choices on the computational overhead associated with drift detection. It became clear that such cost isn't solely dependent on model size but is significantly influenced by the architecture's design and the strategy employed for extracting embeddings. Vector-based methods demonstrated high efficiency across all tested representations, maintaining a memory requirement of less than 0.05 MB per batch. In contrast, distribution-based methods exhibited considerably higher memory consumption, reaching up to 0.93 MB per batch when applied with PCA. These findings highlight that vector-based metrics offer better scalability and are inherently more suited to real-time applications or environments where resources are limited. Moreover, lightweight architecture and effective summarisation techniques like KLL Sketch can greatly enhance memory efficiency. Unlike regular histograms, which use a fixed amount of memory based on the number of bins and dimensions, KLL Sketch adapts its memory usage according to the data's complexity. It employs an adaptive quantile estimation approach that minimises storage needs by strategically storing the most informative data points at varying detail levels. This technique enables highly efficient distribution-based calculations, such as estimating the Wasserstein distance, even with high-dimensional embeddings where conventional methods may be memory-intensive.

Unlike earlier studies that lacked comprehensive benchmarks of various models and data sets, our research includes a broad spectrum of architectures, including autoregressive language models, encoder-decoder frameworks, masked language models, and hybrid models like DeepFM [5, 7, 34, 20]. This variety provides a more complete evaluation of drift detection strategies across different frameworks, helping to identify optimal configurations for balancing detection precision with resource consumption.

RQ4–RQ8 – Trade-offs in Embedding Compression. PCA and KLL were evaluated as dimensionality reduction techniques for embedding compression. PCA preserved more of the original structure and supported better drift sensitivity in vector-based metrics, although

it incurred higher memory and runtime costs. KLL sketches, on the other hand, were more memory-efficient but lost too much signal for practical use in vector-based metrics. However, KLL sketches performed better in the distributional space than PCA histograms. They produced smoother, more linear responses to drift, making them more suitable for use with distribution-based metrics. These results show that the choice between PCA and KLL depends on the metric used. PCA works better for vector distances, while KLL is more effective for distributional comparisons. The choice should be tailored to the specific application.

KLL sketches likely outperformed PCA histograms for several reasons. KLL sketches are designed to maintain compact representations of distributions with bounded memory usage, adapting their resolution to accurately reflect data distribution. This contrasts with histograms, which require fixed storage and use fixed-width bins that may not effectively capture subtle distribution changes. Additionally, PCA adds an extra layer of information loss before histogram creation, while KLL sketches streamline updates, providing better approximations for distribution metrics like quantiles in streaming situations. This computational efficiency is achieved because KLL sketches focus resolution precisely where the data warrants it, allowing memory and accuracy performance, particularly when computing distribution-based distances such as Wasserstein or MMD for drift detection.

No earlier research has systematically compared the performance of PCA and KLL in the context of embedding compression's impact on drift detection efficiency and effectiveness. This study fills this gap, providing valuable insights that contribute to a more informed selection between these techniques, tailored to operational requirements for optimising drift detection efficiency and effectiveness.

RQ6 – Metric Design and Drift Strength Scaling. Existing research highlights a gap in reliable drift detection across various real-world datasets, underlining the need for consistent evaluation protocols. Our results show that embedding format greatly influences drift detection, more so than the specific metric used, reflecting findings by Cao et al. on anomaly detection. Metrics like Euclidean, Manhattan, and Mahalanobis work well on full embeddings but lose effectiveness when used on compressed formats. This highlights the importance of preserving representation quality for reliable results. Distribution metrics like KL divergence and Wasserstein distance excel in detecting drift, especially in detailed, sketch-based formats. This suggests that a chosen embedding technique is vital for effective drift detection, supporting the need for interpretability as stated by the paper of Hinder, Vaquet, and Hammer.

Limitations and Considerations. Several limitations require careful consideration when evaluating the performance of drift detection metrics, especially in low-sample and low-dimensionality scenarios. Figure 7.1 shows a setting with a small corpus of only 100 texts distributed over varying drift strengths, using just two principal components ($n_{\text{components}} = 2$) and a minimal sketch size ($k_{\text{KLL}} = 8$). Under these constraints, vector-based metrics, especially

those derived from KLL summaries, tend to highlight individual outliers aggressively. This often results in spurious spikes that could be misinterpreted as drift. When paired with specific metrics like cosine or Canberra, PCA-based vector distances show instability and inconsistent behaviour, suggesting that their reliability may degrade under severe dimensionality reduction. Conversely, distribution-based metrics demonstrate more predictable trends under the same constraints. KLL, when used in a distributional context, remains comparatively stable even at low sample sizes. PCA, however, again shows varying responses, particularly when applied to histogram-based representations, and fails to track drift consistently. This discrepancy underscores a broader pattern observed throughout this work.

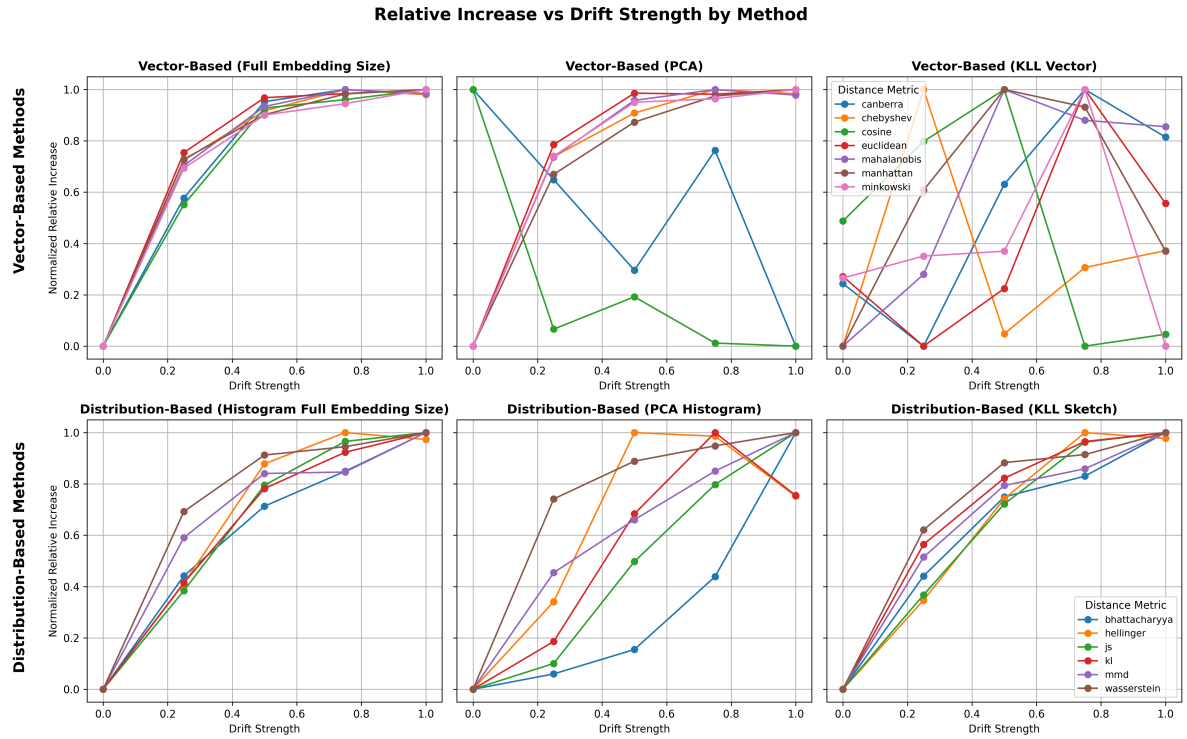


Figure 7.1: With a small corpus of 100 texts—distributed over drift strengths 0:1—and a very low number of principal components ($n_{\text{components}} = 2$) and nearest neighbours ($k_{\text{KLL}} = 8$).

Vector-based metrics are inherently more sensitive and reactive to subtle changes in the embedding space. While this heightened sensitivity enables early warnings, it also increases susceptibility to noise and false positives, especially before any true drift occurs. In real-world data (see Figure 6.4), these metrics often show elevated scores well before any measurable performance degradation, with only a subset maintaining stability in the pre-drift window. In contrast, distribution-based metrics generally exhibit more conservative behaviour. They tend to remain low and stable until actual drift sets in, only rising once a substantive shift is detected. Notable exceptions include the Maximum Mean Discrepancy and Wasserstein distances, which occasionally respond earlier but still do so more selectively than their vector-based counterparts.

This provides a trade-off between early warning and temporal stability. Aggregated vector distances tend to flag drift sooner, offering potentially valuable lead time in dynamic

environments. However, this comes at the cost of more significant variance and more noise in periods of relative data stability. While slower to react, distribution-based metrics provide smoother, more robust signals with fewer false alarms. These limitations suggest that the choice between vector and distribution-based metrics should be guided by operational tolerance for early false positives versus delayed but more reliable detection. This is particularly relevant in settings with tight memory or dimensionality constraints, or where data arrives in small batches. Vector-based metrics are unstable and need to be balanced with the more reliable and steady distribution-based approaches.

Chapter 8

Conclusion

This thesis evaluated the effectiveness and practicality of embedding-based drift detection in dynamic machine learning systems, with a focus on distinguishing between harmful concept drift and benign representation shifts. This study includes a longitudinal evaluation of time-stamped production data from a deployed Amazon recommendation system, empirically comparing vector-based and distribution-based drift metrics, their associated compression techniques, and their performance under various operational constraints.

Despite their susceptibility to noise, the results consistently demonstrate that vector-based metrics are highly effective in detecting performance-impacting drift. In controlled simulations and an eight-year analysis of a production recommender system, vector distances exhibited clear preemptive signals ahead of measurable performance decline. On average, they flagged drift nearly three months before any drop in AUC, offering substantial lead time for intervention. This finding addresses the central research question and positions vector metrics as valuable early-warning tools in non-stationary environments. At the same time, distribution-based metrics demonstrated superior temporal stability, rising more proportionally with the severity of drift and producing smoother signals over time. While slower to react, these metrics proved better suited for long-term monitoring and post-hoc analysis of systemic shifts, especially in settings where false alarms incur operational costs or trigger conservative responses.

The study also shows that embedding the representation format has a greater influence on drift sensitivity than the choice of distance metric itself. Full embeddings preserve semantic detail and support strong signal fidelity but come at the cost of increased memory and computation. Dimensionality reduction techniques, such as PCA and KLL Sketch, offer a spectrum of trade-offs: PCA preserves more structure at a moderate overhead, while KLL offers substantial memory savings with more aggressive information loss, especially in vector-based detection. Vector-based methods also proved more computationally efficient across all embedding formats, consistently requiring less memory and runtime than their distribution-based counterparts. This scalability makes them appropriate for high-frequency monitoring or deployment on resource-constrained systems. These findings contribute a practical and interpretable framework for embedding-based drift detection. The proposed `EmbeddingTracker` pipeline is modular, real-time, and adaptable to various model architectures and deployment contexts. It enables practitioners to calibrate their detection strategy according to specific trade-offs between responsiveness, robustness, and computational cost.

The research addressed several gaps in understanding the application of embedding-based drift detection metrics. It clarified the dynamics of vector-based metrics in anticipating performance-impacting drift, contrasting with traditional methods that primarily react post-degradation. This positions vector-based metrics as essential tools for early intervention strategies, especially in dynamic and non-stationary environments. Furthermore, the analysis underlined the impact of embedding representation formats on drift sensitivity, addressing a previously unexplored dimension where representation fidelity can overshadow the choice of distance metric for drift detection efficacy. By systematically comparing PCA and KLL Sketch, the study reveals their strengths and weaknesses under different dimension reduction constraints, thereby bridging a gap not previously addressed by earlier work. Additionally, the thesis highlighted the scalability of these methods across diverse contexts, providing insight into the cost versus performance benefits that aid practitioners in selecting suitable detection strategies. Through these insights, the study offers a framework for optimising drift detection processes, responding directly to the nuanced needs of modern, resource-constrained systems and contributing significantly to advancing the field of embedding-based drift detection.

8.1 Implications and Future Directions

The observed results and architectural insights from this work lay the groundwork for integrating embedding-based drift detection into real-world machine learning operations. One promising direction is the development of a feedback-driven retraining loop that pairs the responsiveness of vector-based alerts with the reliability of distribution-based validation. As demonstrated in this thesis, vector distances can act as low-latency canary signals, computed on short windows (e.g., hourly PCA-reduced batches), while distribution-based metrics, such as KL divergence, provide high-confidence validation signals over longer horizons (e.g., daily histogram snapshots).

We suggest a drift-aware MLOps pipeline with a shadow model trained on a rolling 90-day window, using exponential decay, to apply these insights. The model is automatically promoted when drift exceeds a sustained threshold and its validation performance surpasses that of the necessary. This system incorporates traffic-split staging, diagnostic Shapley analysis for feature attribution under shift, and scheduled full retraining to accommodate slow-changing seasonal dynamics. Table **??** summarises projected improvements, including reduced detection and recovery times. KLL Sketch helps summarise large logs or telemetry data to estimate latency percentiles without storing all raw events. We can compare these results with shifts in telemetry data.

Figures 8.1 through 8.3 show how this loop integrates with existing MLOps workflows. Drift metrics are surfaced via a real-time observability stack, feeding into automated retraining pipelines. Promotion gates enforce stability through rollback mechanisms, canary deployments, and performance thresholds. This architecture ensures that drift detection is not merely reactive, but embedded in a closed feedback loop, interpretable, auditable, and scalable.

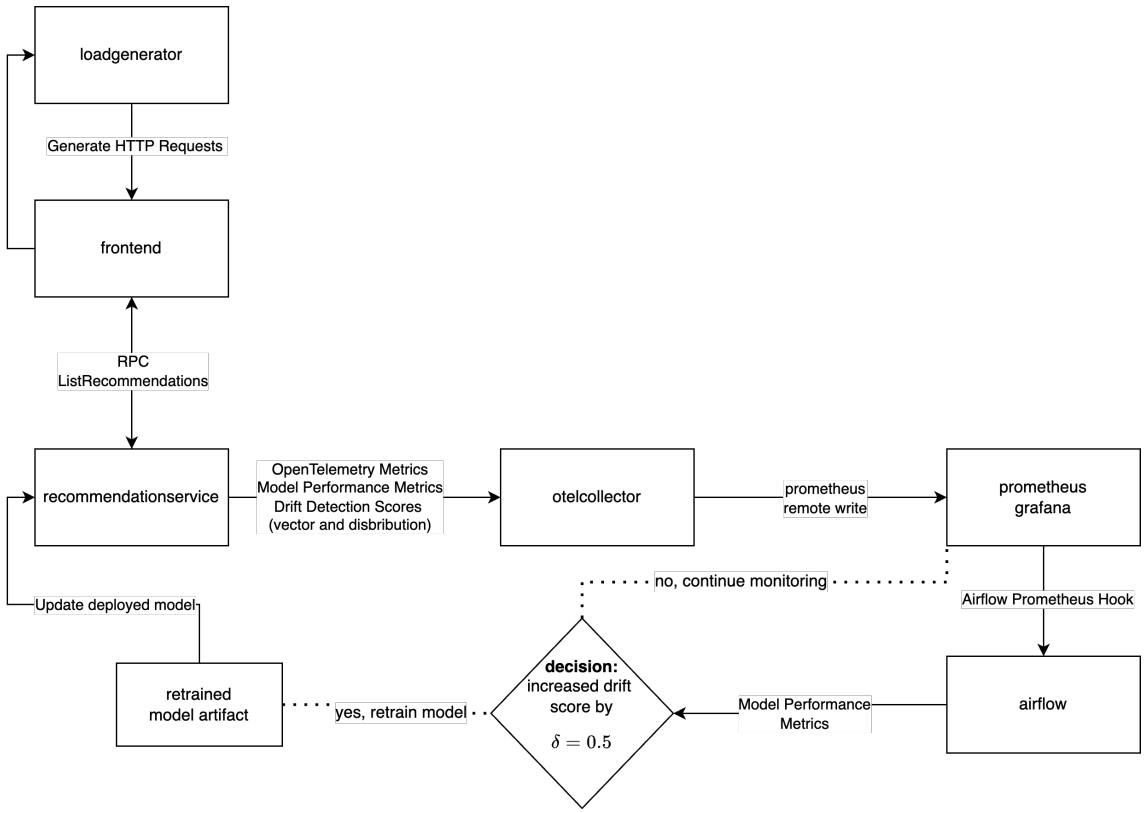


Figure 8.1: Real-time drift detection and retraining trigger architecture. Metric streams and control flows illustrate how distributional shift propagates through the observability stack to initiate model updates.

In conclusion, this thesis demonstrates that embedding-based drift detection is feasible and effective when paired with appropriate representations, compression techniques, and distance metrics. By bridging theoretical sensitivity with operational robustness, the proposed framework enhances the design of resilient machine learning systems that can adapt intelligently to continuous data evolution.

[1] https://github.com/GoogleCloudPlatform/microservices-demo

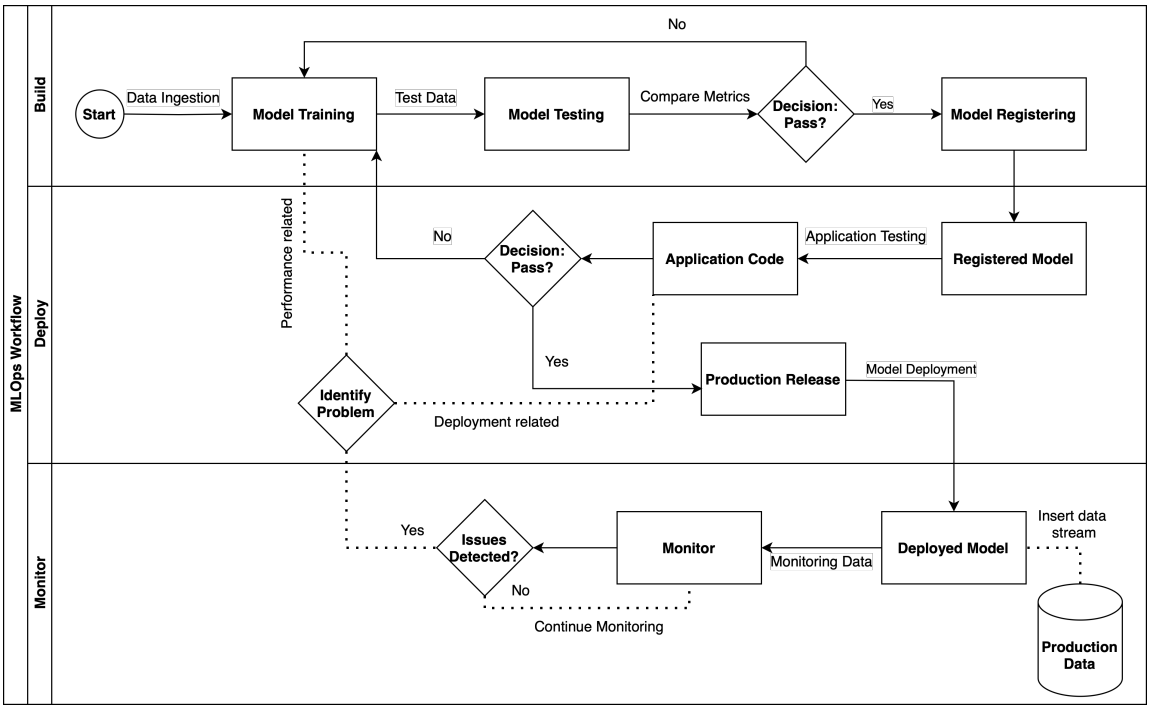


Figure 8.2: Canonical MLOps lifecycle comprising build, deploy and monitor phases. Solid arrows denote the forward progression, while dashed arrows indicate remediation paths activated by performance alerts.

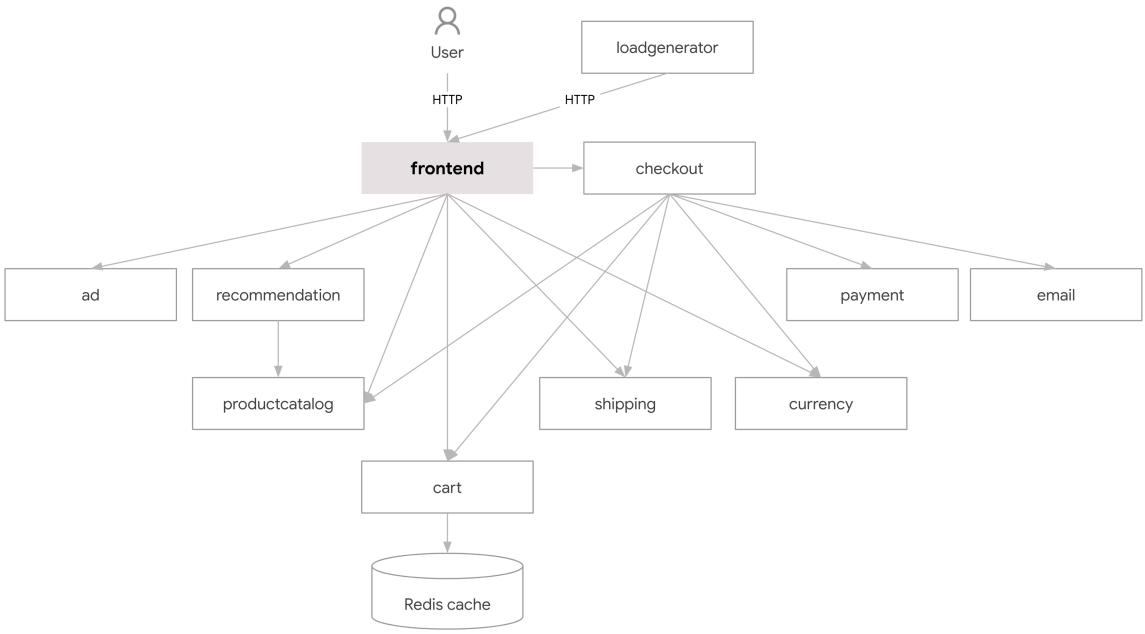


Figure 8.3: Google Cloud Platform's Online Boutique microservices architecture[1]. Machine-learning components interface with traditional services through well-defined APIs, enabling the drift-detection loop to harvest inference telemetry with minimal intrusion.

References

[1] Samuel Ackerman et al. *Theory and Practice of Quality Assurance for Machine Learning Systems An Experiment Driven Approach*. arXiv:2201.00355 [cs]. Apr. 2022. DOI: 10.48550/arXiv.2201.00355. URL: http://arxiv.org/abs/2201.00355 (visited on 04/07/2025).

[2] Supriya Agrahari and Anil Kumar Singh. "Concept Drift Detection in Data Stream Mining : A literature review". In: *Journal of King Saud University - Computer and Information Sciences* 34.10, Part B (Nov. 2022), pp. 9523–9540. ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2021.11.006. URL: https://www.sciencedirect.com/science/article/pii/S1319157821003062 (visited on 12/06/2024).

[3] Sultan Alshamrani. "Distance Matters: Euclidean Embedding Distances for Improved Language Model Generalization and Adaptability". In: *IEEE Access* 12 (2024), pp. 103583–103593. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3434612. URL: https://ieeexplore.ieee.org/abstract/document/10613752 (visited on 04/06/2025).

[4] Raman Arora et al. "Stochastic optimization for PCA and PLS". en. In: *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Monticello, IL, USA: IEEE, Oct. 2012, pp. 861–868. ISBN: 978-1-4673-4539-2 978-1-4673-4537-8 978-1-4673-4538-5. DOI: 10.1109/Allerton.2012.6483308. URL: http://ieeexplore.ieee.org/document/6483308/ (visited on 04/11/2025).

[5] Md Ahsan Ayub and Subhabrata Majumdar. *Embedding-based classifiers can detect prompt injection attacks*. arXiv:2410.22284 [cs]. Oct. 2024. DOI: 10.48550/arXiv.2410.22284. URL: http://arxiv.org/abs/2410.22284 (visited on 04/20/2025).

[6] Eduardo V. L. Barboza et al. *Distance Functions and Normalization Under Stream Scenarios*. arXiv:2307.00106 [cs]. July 2023. DOI: 10.48550/arXiv.2307.00106. URL: http://arxiv.org/abs/2307.00106 (visited on 04/01/2025).

[7] Sidahmed Benabderrahmane et al. *APT-LLM: Embedding-Based Anomaly Detection of Cyber Advanced Persistent Threats Using Large Language Models*. arXiv:2502.09385 [cs]. Feb. 2025. DOI: 10.48550/arXiv.2502.09385. URL: http://arxiv.org/abs/2502.09385 (visited on 04/20/2025).

[8] Albert Bifet. "Classifier Concept Drift Detection and the Illusion of Progress". en. In: *Artificial Intelligence and Soft Computing*. Ed. by Leszek Rutkowski et al. Cham: Springer International Publishing, 2017, pp. 715–725. ISBN: 978-3-319-59060-8. DOI: 10.1007/978-3-319-59060-8_64.

[9] Albert Bifet and Ricard Gavaldà. “Learning from Time-Changing Data with Adaptive Windowing”. In: vol. 7. Apr. 2007. DOI: 10.1137/1.9781611972771.42.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. en. 2006. URL: https://link.springer.com/book/9780387310732 (visited on 05/28/2025).

[11] Dariusz Brzezinski and Jerzy Stefanowski. “Reacting to Different Types of Concept Drift: The Accuracy Updated Ensemble Algorithm”. In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (Jan. 2014). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 81–94. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2013.2251352. URL: https://ieeexplore.ieee.org/document/6494309 (visited on 11/13/2024).

[12] Dariusz Brzeziński and Jerzy Stefanowski. “Accuracy Updated Ensemble for Data Streams with Concept Drift”. en. In: *Hybrid Artificial Intelligent Systems*. Ed. by Emilio Corchado, Marek Kurzyński, and Micha l Woźniak. Berlin, Heidelberg: Springer, 2011, pp. 155–163. ISBN: 978-3-642-21222-2. DOI: 10.1007/978-3-642-21222-2_19.

[13] Yang Cao et al. *TAD-Bench: A Comprehensive Benchmark for Embedding-Based Text Anomaly Detection*. arXiv:2501.11960 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.11960. URL: http://arxiv.org/abs/2501.11960 (visited on 04/19/2025).

[14] Hervé Cardot and David Degras. *Online Principal Component Analysis in High Dimension: Which Algorithm to Choose?* arXiv:1511.03688 [stat]. Nov. 2015. DOI: 10.48550/arXiv.1511.03688. URL: http://arxiv.org/abs/1511.03688 (visited on 04/20/2025).

[15] Rodolfo C Cavalcante and Adriano LI Oliveira. “An approach to handle concept drift in financial time series based on Extreme Learning Machines and explicit Drift Detection”. In: IEEE, 2015, pp. 1–8. ISBN: 1-4799-1960-8.

[16] Emanuele Cavenaghi et al. “Non Stationary Multi-Armed Bandit: Empirical Evaluation of a New Concept Drift-Aware Algorithm”. en. In: *Entropy* 23.3 (Mar. 2021). Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, p. 380. ISSN: 1099-4300. DOI: 10.3390/e23030380. URL: https://www.mdpi.com/1099-4300/23/3/380 (visited on 09/19/2024).

[17] Jaime Céspedes Sisniega et al. “Efficient and scalable covariate drift detection in machine learning systems with serverless computing”. In: *Future Generation Computer Systems* 161 (Dec. 2024), pp. 174–188. ISSN: 0167-739X. DOI: 10.1016/j.future.2024.07.010. URL: https://www.sciencedirect.com/science/article/pii/S0167739X24003716 (visited on 04/07/2025).

[18] Chiao-Ting Chen et al. “Credit Card Fraud Detection via Intelligent Sampling and Self-supervised Learning”. In: *ACM Trans. Intell. Syst. Technol.* 15.2 (Mar. 2024), 35:1–35:29. ISSN: 2157-6904. DOI: 10.1145/3641283. URL: https://dl.acm.org/doi/10.1145/3641283 (visited on 11/15/2024).

[19] Jiaoyan Chen et al. “Knowledge graph embeddings for dealing with concept drift in machine learning”. In: *Journal of Web Semantics* 67 (Feb. 2021), p. 100625. ISSN: 1570-8268. DOI: 10.1016/j.websem.2020.100625. URL: https://www.sciencedirect.com/science/article/pii/S1570826820300585 (visited on 04/19/2025).

[20] Zhi Chen et al. “Is It Overkill? Analyzing Feature-Space Concept Drift in Malware Detectors”. en. In: *2023 IEEE Security and Privacy Workshops (SPW)*. San Francisco, CA, USA: IEEE, May 2023, pp. 21–28. ISBN: 979-8-3503-1236-2. DOI: 10.1109/SPW59333.2023.00007. URL: https://ieeexplore.ieee.org/document/10188641/ (visited on 04/02/2025).

[21] Lénaïc Chizat et al. “Faster Wasserstein Distance Estimation with the Sinkhorn Divergence”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 2257–2269. URL: https://proceedings.neurips.cc/paper/2020/hash/17f98ddf040204eda0af36a108cbdea4-Abstract.html (visited on 04/07/2025).

[22] Per-Erik Danielsson. “Euclidean distance mapping”. In: *Computer Graphics and Image Processing* 14.3 (Nov. 1980), pp. 227–248. ISSN: 0146-664X. DOI: 10.1016/0146-664X(80)90054-4. URL: https://www.sciencedirect.com/science/article/pii/0146664X80900544 (visited on 04/06/2025).

[23] R. De Maesschalck, D. Jouan-Rimbaud, and D. L. Massart. “The Mahalanobis distance”. In: *Chemometrics and Intelligent Laboratory Systems* 50.1 (Jan. 2000), pp. 1–18. ISSN: 0169-7439. DOI: 10.1016/S0169-7439(99)00047-7. URL: https://www.sciencedirect.com/science/article/pii/S0169743999000477 (visited on 04/06/2025).

[24] Ryan Elwell and Robi Polikar. “Incremental Learning of Concept Drift in Nonstationary Environments”. In: *IEEE Transactions on Neural Networks* 22.10 (Oct. 2011). Conference Name: IEEE Transactions on Neural Networks, pp. 1517–1531. ISSN: 1941-0093. DOI: 10.1109/TNN.2011.2160459. URL: https://ieeexplore.ieee.org/document/5975223/?arnumber=5975223 (visited on 09/19/2024).

[25] Tim van Erven and Peter Harremos. “Rényi Divergence and Kullback-Leibler Divergence”. In: *IEEE Transactions on Information Theory* 60.7 (July 2014), pp. 3797–3820. ISSN: 1557-9654. DOI: 10.1109/TIT.2014.2320500. URL: https://ieeexplore.ieee.org/document/6832827/ (visited on 04/07/2025).

[26] M Faisal, E M Zamzami, and Sutarman. “Comparative Analysis of Inter-Centroid K-Means Performance using Euclidean Distance, Canberra Distance and Manhattan Distance”. en. In: *Journal of Physics: Conference Series* 1566.1 (June 2020). Publisher: IOP Publishing, p. 012112. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1566/1/012112. URL: https://dx.doi.org/10.1088/1742-6596/1566/1/012112 (visited on 04/06/2025).

[27] Maxime Fuccellaro. “Concept Drift: detection, update and correction”. fr. In: (2024).

[28] B. Fuglede and F. Topsoe. “Jensen-Shannon divergence and Hilbert space embedding”. In: *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.* June 2004, pp. 31–. DOI: 10.1109/ISIT.2004.1365067. URL: https://ieeexplore.ieee.org/abstract/document/1365067 (visited on 04/07/2025).

[29] João Gama et al. “A survey on concept drift adaptation”. In: *ACM Comput. Surv.* 46.4 (Mar. 2014), 44:1–44:37. ISSN: 0360-0300. DOI: 10.1145/2523813. URL: https://dl.acm.org/doi/10.1145/2523813 (visited on 11/12/2024).

[30] João Gama et al. “Learning with Drift Detection”. en. In: *Advances in Artificial Intelligence – SBIA 2004*. Ed. by Ana L. C. Bazzan and Sofiane Labidi. Berlin, Heidelberg: Springer, 2004, pp. 286–295. ISBN: 978-3-540-28645-5. DOI: 10.1007/978-3-540-28645-5_29.

[31] Hamid Ghorbani. “MAHALANOBIS DISTANCE AND ITS APPLICATION FOR DETECTING MULTIVARIATE OUTLIERS”. en. In: *Facta Universitatis, Series: Mathematics and Informatics* 0 (Oct. 2019). Number: 0, pp. 583–595. ISSN: 2406-047X. DOI: 10.22190/FUMI1903583G. URL: https://casopisi.junis.ni.ac.rs/index.php/FUMathInf/article/view/5028 (visited on 04/06/2025).

[32] Víctor González-Castro, Rocío Alaiz-Rodríguez, and Enrique Alegre. “Class distribution estimation based on the Hellinger distance”. In: *Information Sciences* 218 (Jan. 2013), pp. 146–164. ISSN: 0020-0255. DOI: 10.1016/j.ins.2012.05.028. URL: https://www.sciencedirect.com/science/article/pii/S0020025512004069 (visited on 04/07/2025).

[33] François Goudail, Philippe Réfrégier, and Guillaume Delyon. “Bhattacharyya distance as a contrast parameter for statistical processing of noisy optical images”. EN. In: *JOSA A* 21.7 (July 2004). Publisher: Optica Publishing Group, pp. 1231–1240. ISSN: 1520-8532. DOI: 10.1364/JOSAA.21.001231. URL: https://opg.optica.org/josaa/abstract.cfm?uri=josaa-21-7-1231 (visited on 04/07/2025).

[34] Misgina Tsighe Hagos et al. *Distance-Aware eXplanation Based Learning*. arXiv:2309.05548 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2309.05548. URL: http://arxiv.org/abs/2309.05548 (visited on 04/20/2025).

[35] Ahsanul Haque, Latifur Khan, and Michael Baron. "SAND: Semi-Supervised Adaptive Novel Class Detection and Classification over Data Stream". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (Feb. 2016). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v30i1.10283. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10283 (visited on 11/13/2024).

[36] Fabian Hinder, Valerie Vaquet, and Barbara Hammer. "One or two things we know about concept drift—a survey on monitoring in evolving environments. Part B: locating and explaining concept drift". In: *Frontiers in Artificial Intelligence* 7 (July 2024), p. 1330258. ISSN: 2624-8212. DOI: 10.3389/frai.2024.1330258. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC11294200/ (visited on 04/19/2025).

[37] Yupeng Hou et al. *Bridging Language and Items for Retrieval and Recommendation*. arXiv:2403.03952. Mar. 2024. DOI: 10.48550/arXiv.2403.03952. URL: http://arxiv.org/abs/2403.03952 (visited on 11/14/2024).

[38] Don H Johnson and Sinan Sinanovic. "Symmetrizing the Kullback-Leibler Distance". en. In: (2021).

[39] T. Kailath. "The Divergence and Bhattacharyya Distance Measures in Signal Selection". In: *IEEE Transactions on Communication Technology* 15.1 (Feb. 1967), pp. 52–60. ISSN: 2162-2175. DOI: 10.1109/TCOM.1967.1089532. URL: https://ieeexplore.ieee.org/abstract/document/1089532 (visited on 04/07/2025).

[40] Zohar Karnin, Kevin Lang, and Edo Liberty. *Optimal Quantile Approximation in Streams*. arXiv:1603.05346 [cs]. Apr. 2016. DOI: 10.48550/arXiv.1603.05346. URL: http://arxiv.org/abs/1603.05346 (visited on 04/06/2025).

[41] J Zico Kolter and Marcus A Maloof. "Dynamic weighted majority: An ensemble method for drifting concepts". In: *The Journal of Machine Learning Research* 8 (2007). Publisher: JMLR. org, pp. 2755–2790. ISSN: 1532-4435.

[42] Ali Kore et al. "Empirical data drift detection experiments on real-world medical imaging data". en. In: *Nature Communications* 15.1 (Feb. 2024). Publisher: Nature Publishing Group, p. 1887. ISSN: 2041-1723. DOI: 10.1038/s41467-024-46142-w. URL: https://www.nature.com/articles/s41467-024-46142-w (visited on 12/06/2024).

[43] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (Mar. 1951). Publisher: Institute of Mathematical Statistics, pp. 79–86. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729694. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-1/On-Information-and-Sufficiency/10.1214/aoms/1177729694.full (visited on 12/07/2024).

[44] Cheuk Ting Li, Jingwei Zhang, and Farzan Farnia. "On Convergence in Wasserstein Distance and f-divergence Minimization Problems". en. In: *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. ISSN: 2640-3498. PMLR, Apr. 2024, pp. 2062–2070. URL: https://proceedings.mlr.press/v238/ting-li24a.html (visited on 04/07/2025).

[45] Anjin Liu, Jie Lu, and Guangquan Zhang. "Concept Drift Detection via Equal Intensity k-Means Space Partitioning". In: *IEEE Transactions on Cybernetics* 51.6 (June 2021). Conference Name: IEEE Transactions on Cybernetics, pp. 3198–3211. ISSN: 2168-2275. DOI: 10.1109/TCYB.2020.2983962. URL: https://ieeexplore.ieee.org/abstract/document/9076305 (visited on 11/13/2024).

[46] Jie Lu et al. "Learning under Concept Drift: A Review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (Dec. 2019). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 2346–2363. ISSN: 1558-2191. DOI: 10.1109/TKDE.2018.2876857. URL: https://ieeexplore.ieee.org/abstract/document/8496795 (visited on 11/12/2024).

[47] Daniel Lukats et al. "A benchmark and survey of fully unsupervised concept drift detectors on real-world data streams". en. In: *International Journal of Data Science and Analytics* 19.1 (Jan. 2025), pp. 1–31. ISSN: 2364-4168. DOI: 10.1007/s41060-024-00620-y. URL: https://doi.org/10.1007/s41060-024-00620-y (visited on 04/19/2025).

[48] Neelu Madan et al. "Self-Supervised Masked Convolutional Transformer Block for Anomaly Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.1 (Jan. 2024). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 525–542. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2023.3322604. URL: https://ieeexplore.ieee.org/abstract/document/10273635 (visited on 10/19/2024).

[49] Saranya Maneeroj and Nakarin Sritrakool. "An End-to-End Personalized Preference Drift Aware Sequential Recommender System With Optimal Item Utilization". In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 62932–62952. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3182390. URL: https://ieeexplore.ieee.org/document/9794734/?arnumber=9794734 (visited on 09/19/2024).

[50] Mohammad Masud et al. "Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints". In: *IEEE Transactions on Knowledge and Data Engineering* 23.6 (June 2011). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 859–874. ISSN: 1558-2191. DOI: 10.1109/TKDE.2010.61. URL: https://ieeexplore.ieee.org/abstract/document/5453372 (visited on 11/13/2024).

[51] M. L. Menéndez et al. “The Jensen-Shannon divergence”. In: *Journal of the Franklin Institute* 334.2 (Mar. 1997), pp. 307–318. ISSN: 0016-0032. DOI: 10.1016/S0016-0032(96)00063-4. URL: https://www.sciencedirect.com/science/article/pii/S0016003296000634 (visited on 04/07/2025).

[52] Michael Muhlbaier and Robi Polikar. “Multiple Classifiers Based Incremental Learning Algorithm for Learning in Nonstationary Environments”. In: vol. 6. Sept. 2007, pp. 3618–3623. ISBN: 978-1-4244-0973-0. DOI: 10.1109/ICMLC.2007.4370774.

[53] Soumyaroop Nandi. “ROBUST OBJECT TRACKING AND ADAPTIVE DETECTION FOR AUTO NAVIGATION OF UNMANNED AERIAL VEHICLE”. en. In: ().

[54] Frank Nielsen. “On a Generalization of the Jensen–Shannon Divergence and the Jensen–Shannon Centroid”. en. In: *Entropy* 22.2 (Feb. 2020). Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 221. ISSN: 1099-4300. DOI: 10.3390/e22020221. URL: https://www.mdpi.com/1099-4300/22/2/221 (visited on 04/07/2025).

[55] E. S. Page. “Continuous Inspection Schemes”. In: *Biometrika* 41.1/2 (1954). Publisher: [Oxford University Press, Biometrika Trust], pp. 100–115. ISSN: 0006-3444. DOI: 10.2307/2333009. URL: https://www.jstor.org/stable/2333009 (visited on 11/13/2024).

[56] John Paparrizos et al. *A Survey on Time-Series Distance Measures*. arXiv:2412.20574 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2412.20574. URL: http://arxiv.org/abs/2412.20574 (visited on 04/01/2025).

[57] R. Polikar et al. “Learn++: an incremental learning algorithm for supervised neural networks”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31.4 (Nov. 2001). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), pp. 497–508. ISSN: 1558-2442. DOI: 10.1109/5326.983933. URL: https://ieeexplore.ieee.org/document/983933 (visited on 11/13/2024).

[58] Aniq Ur Rahman, Gourab Ghatak, and Antonio De Domenico. “An Online Algorithm for Computation Offloading in Non-Stationary Environments”. In: *IEEE Communications Letters* 24.10 (Oct. 2020). Conference Name: IEEE Communications Letters, pp. 2167–2171. ISSN: 1558-2558. DOI: 10.1109/LCOMM.2020.3004523. URL: https://ieeexplore.ieee.org/document/9123932/?arnumber=9123932 (visited on 09/19/2024).

[59] Muhammad Rajabinasab et al. “Randomized PCA forest for approximate k-nearest neighbor search”. In: *Expert Systems with Applications* 281 (July 2025), p. 126254. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2024.126254. URL: https://www.sciencedirect.com/science/article/pii/S095741742403121X (visited on 04/20/2025).

[60] É. O. Rodrigues. “Combining Minkowski and Chebyshev: New distance proposal and survey of distance metrics using k-nearest neighbours classifier”. In: *Pattern Recognition Letters* 110 (July 2018), pp. 66–71. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2018.03.021. URL: https://www.sciencedirect.com/science/article/pii/S0167865518301004 (visited on 04/06/2025).

[61] Adam Roe et al. *Semantic Drift Mitigation in Large Language Model Knowledge Retention Using the Residual Knowledge Stability Concept*. en. Nov. 2024. DOI: 10.36227/techrxiv.173091142.28945162/v1. URL: https://www.techrxiv.org/users/848561/articles/1235973-semantic-drift-mitigation-in-large-language-model-knowledge-retention-using-the-residual-knowledge-stability-concept?commit=6c1f1f78ce6c915229fbb66806704a19d2f06e54 (visited on 12/11/2024).

[62] Ylenia Rotalinti et al. “Detecting drift in healthcare AI models based on data availability”. In: Springer, 2022, pp. 243–258.

[63] Tegjyot Singh Sethi and Mehmed Kantardzic. “On the reliable detection of concept drift from streaming unlabeled data”. In: *Expert Systems with Applications* 82 (Oct. 2017), pp. 77–99. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2017.04.008. URL: https://www.sciencedirect.com/science/article/pii/S0957417417302439 (visited on 12/06/2024).

[64] Vinicius M. A. Souza, Farhan A. Chowdhury, and Abdullah Mueen. “Unsupervised Drift Detection on High-speed Data Streams”. en. In: *2020 IEEE International Conference on Big Data (Big Data)*. Atlanta, GA, USA: IEEE, Dec. 2020, pp. 102–111. ISBN: 978-1-7281-6251-5. DOI: 10.1109/BigData50022.2020.9377880. URL: https://ieeexplore.ieee.org/document/9377880/ (visited on 05/28/2025).

[65] Marieke Stolte et al. “Methods for Quantifying Dataset Similarity: a Review, Taxonomy and Comparison”. In: *Statistics Surveys* 18.none (Jan. 2024). arXiv:2312.04078 [stat]. ISSN: 1935-7516. DOI: 10.1214/24-SS149. URL: http://arxiv.org/abs/2312.04078 (visited on 04/07/2025).

[66] R Suwanda, Z Syahputra, and E M Zamzami. “Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K”. en. In: *Journal of Physics: Conference Series* 1566.1 (June 2020). Publisher: IOP Publishing, p. 012058. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1566/1/012058. URL: https://dx.doi.org/10.1088/1742-6596/1566/1/012058 (visited on 04/06/2025).

[67] Arthur Szlam, Yuval Kluger, and Mark Tygert. *An implementation of a randomized algorithm for principal component analysis*. arXiv:1412.3510 [stat]. Dec. 2014. DOI: 10.48550/arXiv.1412.3510. URL: http://arxiv.org/abs/1412.3510 (visited on 04/20/2025).

[68] Volkenandt, Marcel. “Herausforderungen und Chancen von AI Observability durch Integration von ML Modellen in Open Telemetry”. BSc Thesis. Leibniz Universität Hannover, Aug. 2024.

[69] Ke Wan, Yi Liang, and Susik Yoon. “Online Drift Detection with Maximum Concept Discrepancy”. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD ’24. New York, NY, USA: Association for Computing Machinery, Aug. 2024, pp. 2924–2935. ISBN: 979-8-4007-0490-1. DOI: 10.1145/3637528.3672016. URL: https://dl.acm.org/doi/10.1145/3637528.3672016 (visited on 04/19/2025).

[70] Elias Werner et al. *Towards Computational Performance Engineering for Unsupervised Concept Drift Detection – Complexities, Benchmarking, Performance Analysis*. arXiv:2304.08319 [cs]. June 2024. DOI: 10.48550/arXiv.2304.08319. URL: http://arxiv.org/abs/2304.08319 (visited on 04/19/2025).

[71] Jobin Wilson, Santanu Chaudhury, and Brejesh Lall. “Multi-armed bandit based online model selection for concept-drift adaptation”. en. In: *Expert Systems* 41.9 (2024). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/exsy.13626, e13626. ISSN: 1468-0394. DOI: 10.1111/exsy.13626. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13626 (visited on 11/15/2024).

[72] Jiqing Wu et al. “Wasserstein Divergence for GANs”. In: 2018, pp. 653–668. URL: https://openaccess.thecvf.com/content_ECCV_2018/html/Jiqing_Wu_Wasserstein_Divergence_For_ECCV_2018_paper.html (visited on 04/07/2025).

[73] Shiming Xiang, Feiping Nie, and Changshui Zhang. “Learning a Mahalanobis distance metric for data clustering and classification”. In: *Pattern Recognition* 41.12 (Dec. 2008), pp. 3600–3612. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2008.05.018. URL: https://www.sciencedirect.com/science/article/pii/S0031320308002057 (visited on 04/06/2025).

[74] Fuheng Zhao et al. “KLL$^{\pm}$ approximate quantile sketches over dynamic datasets”. en. In: *Proceedings of the VLDB Endowment* 14.7 (Mar. 2021), pp. 1215–1227. ISSN: 2150-8097. DOI: 10.14778/3450980.3450990. URL: https://dl.acm.org/doi/10.14778/3450980.3450990 (visited on 04/07/2025).

[75] Xiulin Zheng et al. “Semi-supervised classification on data streams with recurring concept drift and concept evolution”. In: *Knowledge-Based Systems* 215 (Mar. 2021), p. 106749. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2021.106749. URL: https://www.sciencedirect.com/science/article/pii/S0950705121000125 (visited on 11/13/2024).