High-Dimensional Time Series Anomaly Detection over Heterogeneous Domains

Vincent Jacob

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAX068

Thèse de doctorat

High-Dimensional Time Series Anomaly Detection across Heterogeneous Domains

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique, Données, Intelligence Artificielle

Thèse présentée et soutenue à Palaiseau, le 26 Septembre 2024, par

VINCENT JACOB

Composition du Jury :

Laure Berti-Equille Research Director, Institut de Recherche pour le Développement (ESPACE-DEV)	Rapporteuse
Thorsten Papenbrock Professor, Philipps-Universität Marburg (Fb12)	Rapporteur
Madalina Fiterau Assistant Professor, UMass Amherst (Information Fusion Lab)	Examinatrice
Themis Palpanas Professor, Université Paris Cité (LIPADE)	Examinateur
Jesse Read Professor, École Polytechnique (LIX)	Examinateur (Président)
Yanlei Diao Professor, École Polytechnique (LIX)	Directrice de thèse
Nesime Tatbul Senior Research Scientist, Intel Labs (PCL) and MIT (CSAIL)	Invitée

High-Dimensional Time Series Anomaly Detection across Heterogeneous Domains

Vincent Jacob

A thesis presented for the degree of
Doctor of Philosophy

École Polytechnique
Institut Polytechnique de Paris

Résumé

L'adoption généralisée des services numériques, ainsi que l'échelle et la complexité de leur fonctionnement, ont rendu les incidents dans les opérations informatiques de plus en plus probables, diversifiés et impactants pour les entreprises de logiciels, créant ainsi le besoin de solutions automatisées pour les prévenir.

Pour répondre à ce besoin, cette thèse se concentre sur le problème de la *détection d'anomalies dans les séries temporelles de grande dimension*, dans lequel un large ensemble de séries temporelles multivariées est généré par la surveillance périodique d'entités de service, et où les "anomalies" sont définies comme les motifs déviant d'une certaine notion de comportement normal. Cette tâche, s'inscrivant dans le domaine émergent de l'"Intelligence Artificielle pour les Opérations Informatiques" (AIOps), présente plusieurs défis clés (ou *challenges*) abordés dans ce travail, à savoir : (**CH0**) *l'absence d'outils de benchmarking ouverts* reflétant les défis de l'AIOps dans la recherche académique, (**CH1**) *la rareté des annotations d'anomalie* disponibles pour entraîner les méthodes de détection, (**CH2**) *la grande dimensionnalité, complexité et variété des comportements normaux* pour les séries temporelles collectées, (**CH3**) *le changement dans les comportements normaux* pouvant survenir entre les données d'entraînement et de test, et (**CH4**) *la présence de motifs de "bruit" normaux mais minoritaires* dans les données, ne devant pas être détectés comme anomalies par les méthodes.

Cette thèse commence par aborder **CH0**, à travers la contribution (**CTB1**) *de nouveaux outils de benchmarking pour la détection d'anomalies explicable dans des séries temporelles de grande dimension*. L'outil principal est le benchmark **Exathlon**, comprenant (i) un jeu de données annoté provenant de la surveillance d'applications Spark Streaming, (ii) une méthodologie d'évaluation flexible de la performance des méthodes de détection d'anomalies (AD) et de "découverte d'explications" (ED), et (iii) une pipeline complète de détection d'anomalies explicable pour faciliter la création et l'évaluation de méthodes. Afin d'évaluer davantage les prédictions des méthodes de détection et d'explication d'anomalies, cette thèse propose également la conception d'une nouvelle plateforme de visualisation appelée LEADS Viewer, fournissant une interface graphique pour visualiser les résultats de détection d'anomalies explicable dans des séries temporelles de grande dimension.

Le reste de cette thèse se concentre sur la composante "détection d'anomalies" du benchmark Exathlon. Pour s'accorder avec **CH1**, il considère principalement des méthodes de détection dans un cadre *non supervisé*, supposant des données d'entraînement non annotées et principalement normales. Dans ce cadre, cette thèse propose (**CTB2**) *une analyse de benchmark approfondie de méthodes représentatives non supervisées* et semi-supervisées avec annotations de données normales, simplement appelées "non supervisées" ici. Notre étude révèle une performance limitée de toutes les méthodes comparées pour résoudre le benchmark Exathlon, avec un score F1 maximal obtenu de seulement 0.66. Elle montre également que cette performance limitée est principalement due à une grande vulnérabilité de ces méthodes à **CH3**, en particulier pour les méthodes d'apprentissage profond. En effet, ces dernières modélisent plus finement les données d'entraînement que les alternatives classiques étant donné **CH2**, ce qui est un avantage en AIOps et motive l'utilisation de l'apprentissage profond, mais est contrebalancé ici par une sensibilité accrue au changement de distribution. Dans l'ensemble, toutes les méthodes ont montré trois **limitations** principales, avec un impact décroissant sur la performance : (**L1**) *une vulnérabilité au changement de comportement normal entre les données d'entraînement et*

de test, (**L2**) *une production de faux négatifs pour les anomalies les plus complexes étant donné notre grand nombre de features*, et (**L3**) *une production de faux positifs pour les motifs de "bruit" normaux mais minoritaires dans les données de test*.

La partie suivante de cette thèse aborde explicitement **L1** via (**CTB3**) *une nouvelle méthode de généralisation de domaine pour la détection d'anomalies non supervisée*, définissant les différents *contextes* de fonctionnement normal comme un ensemble de *domaines hétérogènes*, et associant le changement de comportement normal au concept de *changement de domaine*. Nous commençons par caractériser formellement le problème de la détection d'anomalies non supervisée sous changement de domaine. Nous proposons ensuite "Domain-Invariant VAE for Anomaly Detection", ou **DIVAD**, s'appuyant sur le *feature disentanglement* pour décomposer la variable observée en encodages *spécifique au domaine* et *invariants au domaine*, et définissant les anomalies comme les observations déviant de la distribution d'entraînement des encodages invariants au domaine uniquement. Nous concevons différentes variantes de DIVAD, et montrons que les meilleures d'entre elles sont particulièrement efficaces pour traiter la limitation **L1** de nos méthodes de référence, améliorant ainsi leur performance de détection de manière significative (jusqu'à 20%, atteignant un score F1 maximal de 0.79). Nous appliquons également notre méthode DIVAD au jeu de données "Application Server Dataset", et montrons qu'elle surpasse la meilleure méthode de référence en score F1 maximal pour 92% des cas de test (avec une amélioration de plus de 10% pour 67% d'entre eux), soulignant ainsi son applicabilité plus large, au-delà de notre jeu de données.

Malgré le succès de notre méthode DIVAD pour traiter la principale limitation **L1** de nos méthodes de référence, elle n'a pas été conçue pour s'attaquer à leurs limitations restantes **L2** et **L3**. Pour traiter conjointement ces trois limitations, cette thèse propose (**CTB4**) *de nouvelles méthodes contrastives dans un cadre faiblement supervisée*, considérant quelques instances d'anomalie annotées pour entraîner les méthodes de détection, tout en s'accordant toujours avec **CH1**. Dans ce nouveau cadre, nous proposons "Contrastive Encoder for Anomaly Detection with a Few Anomaly Labels", ou **CEADAL**, une méthode (i) extrayant des *paires* normal-normal et normal-anomalie à partir des données d'entraînement, (ii) s'appuyant sur la *perte contrastive* pour construire une projection latente dans laquelle les données normales sont regroupées dans une région restreinte, éloignée des anomalies, et (iii) définissant les anomalies comme les observations déviant de la distribution d'entraînement des observations normales latentes. Afin d'étudier l'impact de la perte contrastive utilisée par CEADAL, nous proposons également une méthode alternative basée sur la *perte par triplet*, appelée TEADAL, induisant des représentations plus riches et moins contraintes pour les observations normales dans l'espace latent. Nous évaluons CEADAL et TEADAL sur le benchmark Exathlon, et montrons l'efficacité de CEADAL pour traiter nos limitations **L1-3**, atteignant ainsi les scores F1 maximal et médian les plus élevés parmi toutes les méthodes comparées (0.83 et 0.80, respectivement). Notre étude confirme notamment que les fortes contraintes de proximité imposées par CEADAL pour les observations normales latentes tendent à induire une "généralisation de contexte implicite" qui aide à aborder **L1**, surpassant ainsi des alternatives moins contraintes telles que TEADAL (qui n'a obtenu qu'un score F1 maximal de 0.64, malgré une meilleure modélisation des données d'entraînement). Notre étude montre enfin que les quelques anomalies annotées utilisées par CEADAL sont en effet utiles pour traiter **L2** et **L3**, réduisant ainsi le nombre de faux négatifs et de faux positifs produits.

Abstract

The widespread adoption of digital services, along with the scale and complexity at which they operate, has made incidents in IT operations increasingly likely, diverse and impactful for software companies, creating the need for automated methods to prevent them.

To respond to such needs, this thesis focuses on the problem of *anomaly detection in high-dimensional time series*, where a large set of multivariate time series is generated from the periodic monitoring of service entities, and "anomalies" are defined as patterns in data that deviate from a given notion of normal behavior. This task, lying in the emerging field of "Artificial Intelligence for IT Operations" (AIOps), presents a set of key **challenges** addressed in this work, namely: (**CH0**) *the lack of open benchmarking tools* that reflect AIOps challenges in academic research, (**CH1**) *the scarcity of anomaly labels* available to train the detection methods, (**CH2**) *the high dimensionality, complexity and variety of normal behaviors* for the recorded time series, (**CH3**) *the shift in normal behaviors* that can occur from training to test data, and (**CH4**) *the presence of normal but "noisy", minority patterns* in both training and test data that should not be flagged as anomalies by the methods.

This thesis starts by addressing **CH0** through the contribution of (**CTB1**) *new benchmarking tools for explainable anomaly detection in high-dimensional time series*. The main tool is the **Exathlon** benchmark, consisting of (i) a labeled dataset centered around a Spark Streaming application monitoring use case, (ii) a flexible evaluation methodology to assess the performance of anomaly detection (AD) and explanation discovery (ED) methods, and (iii) an end-to-end explainable anomaly detection pipeline to facilitate method building and benchmarking. To further diagnose the predictions of anomaly detection and explanation methods, this thesis also contributes the design of a new visualization platform called LEADS Viewer, providing a Graphical User Interface (GUI) to visualize explainable anomaly detection outputs on high-dimensional time series.

The remainder of this thesis focuses on the anomaly detection component of the Exathlon benchmark. To comply with **CH1**, it mainly considers AD methods in an *unsupervised* setting, assuming unlabeled, mostly-normal data for training. Under this setting, this thesis contributes (**CTB2**) *an in-depth benchmarking analysis of representative unsupervised* and semi-supervised *methods* assuming labeled normal data, simply referred to as "unsupervised" here. Our study reveals the limited performance of all the methods compared in solving the Exathlon benchmark, with a maximum peak F1-score achieved of only 0.66. It also shows that this limited performance is primarily due to a high vulnerability of these methods to **CH3**, especially for deep learning methods. Such methods indeed model training data at a finer level than shallow alternatives given **CH2**, which is an advantage in AIOps and motivates using deep learning, but is counterbalanced here by an increased sensitivity to distribution shift. Overall, all methods displayed three main **limitations**, with a decreasing impact on the performance: (**L1**) *a vulnerability to normal behavior shift from training to test data*, (**L2**) *a production of false negatives for the hardest anomalies given our large number of features*, and (**L3**) *a production of false positives for normal but "noisy", minority patterns in test data*.

The next part of this thesis explicitly addresses **L1** with (**CTB3**) *a new domain generalization method for unsupervised anomaly detection*, defining the different *contexts* of normal operation as a set of *heterogeneous domains*, and associating normal behavior shift to the concept of *domain shift*. We start by formally characterizing the problem of unsu-

pervised anomaly detection under domain shift. We then propose Domain-Invariant VAE for Anomaly Detection, or **DIVAD**, relying on feature disentanglement to decompose the observed variable into *domain-specific* and *domain-invariant* encodings, and defining anomalies as samples that deviate from the training distribution of domain-invariant encodings only. We design different DIVAD variants, and show that the best ones are particularly effective in addressing the limitation **L1** of our unsupervised baselines, thus improving their anomaly detection performance by a large margin (up to 20%, reaching a maximum peak F1-score of 0.79). We also apply our DIVAD method to the Application Server Dataset (ASD), and show that it outperforms the best unsupervised baseline in maximum peak F1-score for 92% of the test cases (with over 10% improvements for 67% of them), hence highlighting its broader applicability beyond our dataset.

Despite the success of DIVAD in addressing the main limitation **L1** of our baselines, it was not designed to tackle their remaining limitations **L2** and **L3**. To jointly address all three limitations, this thesis contributes (**CTB4**) *new contrastive methods in a weakly-supervised setting*, considering a few labeled anomalies to train the detection methods, while maintaining compliance with **CH1**. Under this new setting, we propose Contrastive Encoder for Anomaly Detection with a Few Anomaly Labels, or **CEADAL**, a method that (i) extracts normal-normal and normal-anomaly *pairs* from the training samples, (ii) relies on the *contrastive loss* to construct a latent mapping where normal data samples are grouped within a tight region, away from the anomalies, and (iii) defines anomalies as samples that deviate from the training distribution of normal latent samples. To study the impact of the contrastive loss used by CEADAL, we also propose an alternative method based on the more recent *triplet loss* called TEADAL, inducing richer, less-constrained representations of normal samples in latent space. We apply both CEADAL and TEADAL against the Exathlon benchmark, and show the effectiveness of CEADAL in addressing our limitations **L1-3**, thus achieving the highest maximum and median peak F1-scores across all the methods compared (0.83 and 0.80, respectively). In particular, our study confirms that the strong proximity constraints enforced by CEADAL for normal latent samples tends to induce an "implicit context generalization" that combats **L1**, thus outperforming less constrained alternatives like TEADAL (which obtained a maximum peak F1-score of only 0.64, despite a better modeling of the training data). Our study finally shows that the few labeled anomalies leveraged by CEADAL are indeed helpful in tackling **L2** and **L3**, thereby reducing the amount of false negatives and false positives produced.

Acknowledgments

I am deeply grateful for the support and encouragement I have received during the completion of this PhD thesis.

I would like to start by thanking my supervisor, Yanlei, for her guidance, advice, and continuous support throughout this research. I am also grateful to the reviewers and Jury members of my thesis, Laure Berti-Equille, Thorsten Papenbrock, Madalina Fiterau, Themis Palpanas and Jesse Read, for their time spent evaluating my work and their valuable feedback, as well as to Alexandre Gramfort for his advice and support as a monitoring committee member.

I extend my thanks to all my co-authors and collaborators across research projects, with whom I have been fortunate to work: Nesime, Fei, Bijan and Arnaud for their contributions to Exathlon; Jia Li, Ran Wang and Junqiang Chen for their work on LEADS Viewer; as well as Iman and Sein Minn for their efforts on our fraud detection project. Special thanks also go to all members of the CEDAR team whose paths I have crossed over the years, for the insightful discussions and moments shared, as well as to Jessica and Alice for their administrative assistance.

On a personal note, I would like to deeply thank my family and friends, for their unwavering support, help and encouragement throughout this journey.

Contents

List of Figures

List of Tables

Notation

This work adopts most of the notational conventions of [1], with some exceptions regarding sets and an additional "slicing" notation for vectors and matrices.

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable

Sets

$\mathcal{A}$	A set
$\mathbb{R}$	The set of real numbers
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$[a \,..\, b]$	The integer interval including a and b
$\{a, b, c\}$	The set containing a, b and c
$\{a_1, \ldots, a_n\}$	The set containing $a_1, a_2, \cdots$ up to a_n

Indexing

For all elements, indexing starts at 1.

a_i	Element i of vector $\boldsymbol{a}$
$\boldsymbol{a}_{i:j}$	Elements i to j in vector $\boldsymbol{a}$ (both included)
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$ or $\boldsymbol{A}_i$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i:j}$	Rows i to j in matrix $\boldsymbol{A}$ (both included)
$\boldsymbol{A}_{i:j,k:l}$	Rows i to j and columns k to l in matrix $\boldsymbol{A}$ (all included)

1 Introduction

Over the past few decades, the norm in the software industry has been shifting from periodically releasing products to providing cloud-based *services* [2]. The International Data Corporation (IDC) predicts that worldwide spending on public cloud services will reach $805 billion in 2024 and double by 2028, with applications being the largest category, capturing over 40% of all public cloud spending in 2024 [3]. This paradigm shift has greatly generalized the adoption of DevOps practices [4], aiming to constantly improve service quality by implementing continuous development and release cycles. In this DevOps context, companies increasingly rely on site reliability engineers (SREs) to continually maintain services under operational conditions, in compliance with service-level agreements (SLAs) previously made with customers. As such, incidents in IT operations have become more impactful for software companies, inducing ever-increasing financial costs, both directly through these SLAs and indirectly through brand image deterioration. Concurrently, the popularity of such services has greatly increased the scale and complexity at which they operate, always relying on more resources and parallelism to process larger volumes of data at high speed. This evolution has made incidents more frequent, diverse and difficult for SREs to manually anticipate and diagnose, thus creating the need for more automated solutions.

Providing such solutions has motivated the rapid development of "Artificial Intelligence for IT Operations" (AIOps) [5], more broadly proposing to use AI to automate and optimize operational workflows [6]. The field of AIOps has been gaining significant attention within the past few years, with a market size estimated to grow from $27.24 billion in 2024 to $79.91 billion by 2029 [7], as the community recognizes the critical role it will play in the future of service delivery and IT Operations Management (ITOM). Toward the application goal of providing systems that can effectively assist SREs in preventing service incidents, this thesis focuses on the problem of *anomaly detection in high-dimensional time series*, where a large set of multivariate time series is generated from the periodic monitoring of service *entities*, and "anomalies" are defined as patterns in data that deviate from a given notion of normal behavior [8]. Figure 1.1 shows an illustrative example for a service entity run and corresponding multivariate time series. In this example, the periodic monitoring of the entity's execution generates multiple *univariate* time series with aligned time indices, constituting a *multivariate* time series. The goal is then to detect the anomalous time period (marked in red) from the time series data, corresponding to a period of abnormal behavior for the running entity.

1.1 Technical Challenges

Detecting anomalies in AIOps presents a set of key **challenges** [9] covered in this work, which we organize as follows:

- **CH0.** *The lack of open benchmarking tools* that reflect the AIOps challenges in academic research. This challenge supersedes all others, as it prevents the assessment of whether the developed methods can effectively tackle them.
- **CH1.** *The scarcity of anomaly labels* available to train the detection methods, due to the domain knowledge of IT operations required to reliably label anomalies, and the labor-intensive process of examining large amounts of time series data.

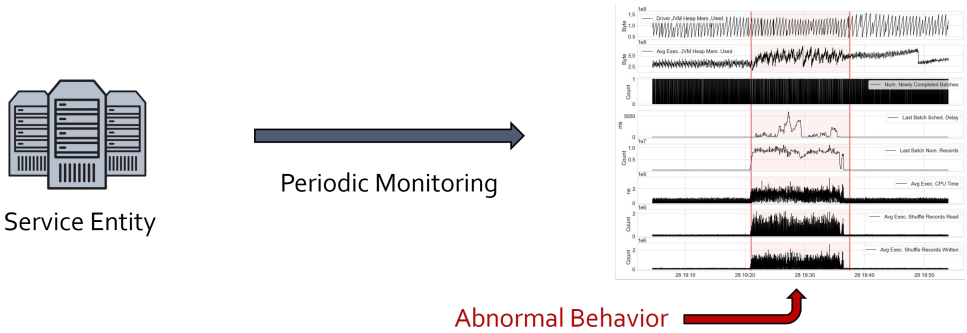


Figure 1.1: The periodic monitoring of a service entity generates a multivariate time series, where anomalous time periods correspond to abnormal behaviors for the running entity.

- **CH2.** *The high dimensionality, complexity and variety of normal behaviors* for the recorded time series. The collection of hundreds or thousands of metrics, monitored at a high frequency for a large number of service entities indeed yields high-dimensional data across both the time and feature dimensions. This high-dimensional data being monitored for multiple, complex entities at scale in different contexts also adds to the complexity and variety of the normal behaviors recorded.
- **CH3.** *The shift in normal behaviors* that can occur from training to test data, due to potentially frequent changes in software components, hardware components or operation contexts of the monitored entities.
- **CH4.** *The presence of normal but "noisy", minority patterns* in both training and test data that should not be flagged as anomalies by the methods, due to the fine granularity of data collection and the multiple sources of random artifacts that can occur.

CH0 has long limited academic research efforts toward anomaly detection in AIOps, with existing anomaly detection (AD) benchmarks and datasets operating on very small scales compared to typical real-world settings [10, 11], while also not reflecting **CH3** in systematic and controlled ways. For this reason, addressing **CH0** constitutes the first focus of this thesis. Despite these benchmarking limitations, a significant amount of time series anomaly detection methods have been proposed over the years. In the literature, **CH1** has been addressed through the development of unsupervised and semi-supervised AD methods [8], assuming no or only partial label information for training, respectively. The advent of deep learning (DL) [12] has also been instrumental in partly addressing **CH2**, with deep methods becoming particularly effective at learning rich representations of high-dimensional data as the number of training samples increases, being able to capture both temporal (i.e., intra-feature) and spatial (i.e., inter-feature) dependencies of multivariate time series [13, 14]. **CH3**, despite being one of the most critical challenges of anomaly detection in AIOps, has been the least studied in the literature. As a result, this thesis identifies the main limitation of current anomaly detection methods as their lack of robustness to a shift in normal behaviors from training to test data, leading them to assign different degrees of "abnormality" to distinct *contexts* of *normal* operation for the recorded entities. Finally, being vulnerable to **CH4** can hinder the performance and practical usage of AD methods, causing them to generate numerous false positives due to an interference between noise and anomalies in test data. As this thesis shows, leveraging a few labeled anomaly instances for training, in a *weakly-supervised* setting [15, 16], can be an effective way to address this challenge while maintaining compliance with **CH1**.

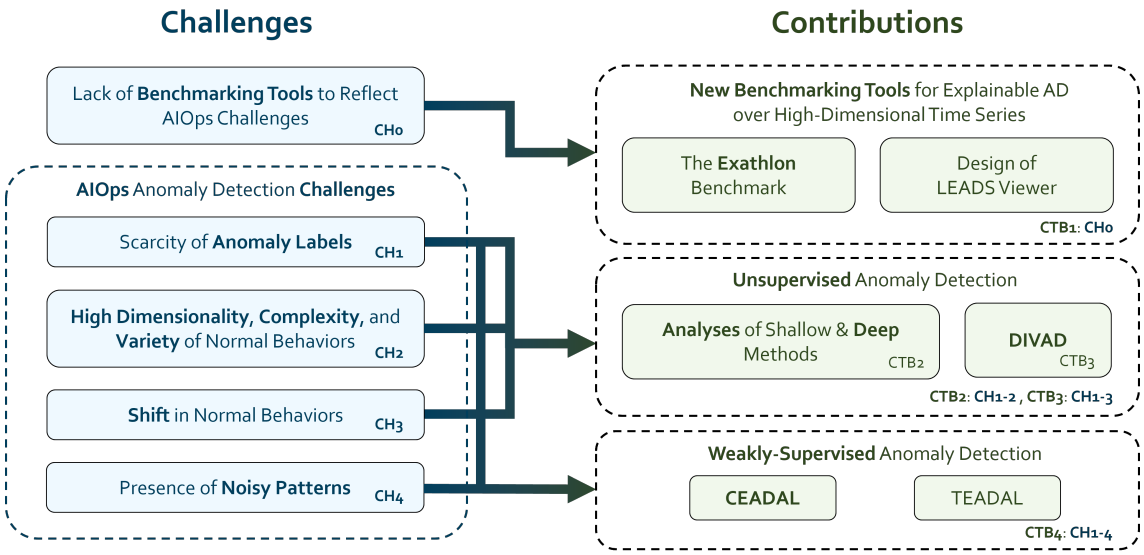


Figure 1.2: Addressed challenges **CH0-4** and corresponding contributions **CTB1-4**.

1.2 Scope and Contributions

This thesis proposes to address the challenges introduced above through four main **contributions** (summarized in Figure 1.2):

CTB1 *New benchmarking tools for explainable anomaly detection in high-dimensional time series* (Chapter 3). To address **CH0**, the main contribution of this thesis is **Exathlon** [17], the first public benchmark for explainable anomaly detection over high-dimensional time series, publicly available at `https://github.com/exathlonbenchmark/exathlon`. Exathlon consists of (i) a labeled dataset centered around a Spark Streaming application monitoring use case, (ii) a flexible evaluation methodology to assess the performance of *anomaly detection* (AD) and *explanation discovery* (ED) methods, and (iii) an end-to-end explainable anomaly detection pipeline to facilitate method building and benchmarking. Through its large-scale distributed stream processing use case, Exathlon's dataset was collected with AIOps considerations in mind, providing data traces that significantly exceed the dimensionality of traditional time series datasets and benchmarks. The design of its evaluation methodology and pipeline is however modular and extensible, making Exathlon applicable to a wide variety of explainable anomaly detection problems beyond this particular use case. Besides anomaly detection, Exathlon supports the evaluation of "explanation discovery", defined as the task of deriving human-readable, meaningful explanations for the anomalies detected (e.g., in the form of feature importance scores or decision rules involving the input features). To further diagnose the predictions of anomaly detection and explanation methods, this thesis also contributes the design of a new visualization platform called LEADS Viewer, providing a Graphical User Interface (GUI) to visualize explainable anomaly detection outputs on high-dimensional time series data, publicly available at `https://github.com/exathlonbenchmark/leads-viewer`.

CTB2 *An in-depth benchmarking analysis of representative unsupervised anomaly detection methods* (Chapter 4). The remainder of this thesis focuses on the anomaly detection component of the Exathlon benchmark. To comply with **CH1**, it mainly considers anomaly detection methods in an *unsupervised* setting, assuming unlabeled, mostly-normal data for training [18]. Under this setting, this thesis contributes an in-depth benchmarking analysis of representative unsupervised and semi-supervised methods assuming labeled

normal data (simply referred to as "unsupervised" in the following), as per the taxonomy of Schmidl et al. [19]. Our study reveals the limited performance of all the methods compared in solving the Exathlon benchmark, with a maximum peak F1-score of only 0.66 achieved by TranAD [14]. It also shows that this limited performance, as well as some of the tradeoffs observed, are mostly due to a high vulnerability of these methods to **CH3**, with shifts in *contexts* for the *normal* operations of Exathlon's Spark Streaming applications being considered abnormal by the methods. This vulnerability to normal behavior shift is especially pronounced for deep learning methods, which is caused by, and counterbalances, their finer modeling of the training data compared to shallow alternatives given **CH2**. Our study finally shows that all the methods compared produce *false negatives* with respect to the hardest anomalies of Exathlon's dataset, while also being vulnerable to **CH4**, producing *false positives* for normal but "noisy", minority patterns in test data. Overall, the methods displayed three main **limitations**, listed below in decreasing order of impact on the AD performance:

- **L1.** *A vulnerability to normal behavior shift from training to test data*, hence a shortcoming in addressing **CH3** above.
- **L2.** *A production of false negatives for the hardest anomalies given our large number of features*, hence a shortcoming in addressing part of **CH2** above.
- **L3.** *A production of false positives for normal but "noisy", minority patterns in test data*, hence a shortcoming in addressing **CH4** above.

CTB3 *A new explicit domain generalization method for unsupervised anomaly detection* (Chapter 5). The next contribution of this thesis is to explicitly address **L1**, identified as the main limitation of our unsupervised baselines. We do so through the scope of *domain generalization*, defining the different *contexts* of normal operation for the recorded entities as a set of *heterogeneous domains*, and associating normal behavior shift to the concept of *domain shift*. We start by formally characterizing the problem of unsupervised anomaly detection under domain shift. We then propose Domain-Invariant VAE for Anomaly Detection, or **DIVAD**, relying on feature disentanglement [20, 21] to decompose the observed variable into *domain-specific* and *domain-invariant* encodings, and defining anomalies as samples that deviate from the training distribution of domain-invariant encodings only. We design different DIVAD variants, based on the architecture, type of domain-invariant encoding prior used (fixed standard Gaussian vs. learned Gaussian Mixture) and anomaly scoring strategy. We show that the best ones are particularly effective in addressing the limitation **L1** of our unsupervised baselines, thus improving their anomaly detection performance by a large margin (up to 20% for the learned Gaussian Mixture variant, reaching a maximum peak F1-score of 0.79). We also apply our DIVAD method to the Application Server Dataset (ASD) [13], assessing its ability to detect anomalies in a test server given training normal data from 11 different servers. Our experiment shows that DIVAD outperforms the best unsupervised baseline TranAD in maximum peak F1-score for 92% of the test cases (with over 10% improvements for 67% of them), hence highlighting its broader applicability beyond our Spark Streaming dataset.

CTB4 *New contrastive methods in a weakly-supervised setting* (Chapter 6). Despite the success of DIVAD in addressing the main limitation **L1** of our baselines, this method was not designed to tackle their remaining limitations **L2** and **L3**. To jointly address all three limitations, this thesis explores a *weakly-supervised* extension of our setting, considering a few labeled anomalies to train the detection methods, while maintaining compliance with **CH1**. Under this new setting, we propose Contrastive Encoder for Anomaly Detection with a Few Anomaly Labels, or **CEADAL**, a method that (i) extracts normal-normal and normal-anomaly *pairs* from the training samples, (ii) relies on the *contrastive loss* [22] to construct a latent mapping where normal data samples are grouped within a tight region,

away from the anomalies, and (iii) defines anomalies as samples that deviate from the training distribution of normal latent samples. To study the impact of the contrastive loss used by CEADAL, we also propose an alternative method based on the more recent *triplet loss* [23] called TEADAL (using normal samples as anchor and positive inputs, and anomalies as negative inputs), inducing richer, less-constrained representations of normal samples in latent space. We apply both CEADAL and TEADAL against the Exathlon benchmark, along with other encoding and classification methods, and show the effectiveness of CEADAL in addressing our limitations **L1-3**, achieving the highest maximum and median peak F1-scores across all the methods compared (0.83 and 0.80, respectively). In particular, our study confirms that the strong proximity constraints enforced by CEADAL for normal latent samples tends to induce an "implicit context generalization" that combats **L1**, thus outperforming less constrained alternatives like TEADAL (which obtained a maximum peak F1-score of only 0.64, despite a better modeling of the training data). Our study finally shows that the few labeled anomalies leveraged by CEADAL are indeed helpful in tackling **L2** and **L3**, thereby reducing the amount of false negatives and false positives produced.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides a review of the relevant literature, including the datasets and benchmarks proposed for time series anomaly detection (AD) and explanation, as well as the unsupervised, weakly-supervised and domain generalization methods proposed for AD in AIOps scenarios. It also introduces and categorizes the main *explanation discovery* (ED) methods mentioned and targeted by the Exathlon benchmark. Chapter 3 presents the Exathlon benchmark, covering its Spark Streaming dataset, evaluation methodology and end-to-end pipeline in detail. It also introduces LEADS Viewer, and the way this platform can be used to diagnose the predictions of anomaly detection and explanation methods. Chapter 4 relies on the Exathlon benchmark to present an experimental study of unsupervised anomaly detection methods, providing additional details on Exathlon's dataset, and highlighting the characteristics, tradeoffs and limitations of the methods compared. Chapter 5 presents DIVAD, a new domain generalization method for unsupervised anomaly detection, showing this method effectively addresses the vulnerability to normal behavior shift identified for the unsupervised baselines, while also being more broadly applicable beyond our particular dataset. Chapter 6 considers a weakly-supervised extension of our setting, where the detection methods can additionally leverage a few labeled anomalies for training. It presents CEADAL and TEADAL, relying on the contrastive loss and triplet loss, respectively, to construct latent representations of normal data samples lying away from anomalies. It shows that CEADAL addresses the main limitations identified for the unsupervised baselines, achieving the best anomaly detection performance overall. Finally, Chapter 7 presents the main conclusions of this thesis and suggests relevant research directions for future work.

2 Literature Review

In this chapter, we survey the existing work that relates to benchmarking and building explainable anomaly detection methods in our AIOps use case. We start in Section 2.1 by reviewing the datasets and benchmarks proposed for time series anomaly detection and explanation before and following Exathlon. We then discuss relevant anomaly detection and explanation methods in Sections 2.2 and 2.3, respectively.

2.1 Datasets and Benchmarks

This section reviews the datasets and benchmarks that have been proposed for time series anomaly detection and explanation.

2.1.1 Anomaly Detection in Multivariate Time Series

Anomaly detection in multivariate time series can be defined as the task of providing accurate anomaly outputs for multivariate *data records* in a set of test *sequences*. These anomaly outputs can take the form of *binary predictions* (e.g., 0 for normal, 1 for anomaly), in which case the goal is to accurately classify normal from anomalous records in test data, or real-valued *anomaly scores*, in which case the goal is to assign higher anomaly scores to anomalous records than to normal records in test data.

Analyzing and comparing anomaly detection methods on time series data has a long history [24, 18, 25]. However, before Exathlon, the only work really released as a public benchmark for time series anomaly detection was the Numenta Anomaly Benchmark (NAB) [10, 26]. This benchmark focuses on methods developed for real-time streaming data. It provides more than 50 real and artificial time series data files, along with a scoring system designed to reward early detection. Although some of these data files cover our main AIOps use case, they are mostly univariate and much smaller in scale than Exathlon's. Further, NAB presents several technical challenges that hinder its use as a practical anomaly detection benchmark [27].

During and since the release of Exathlon, the growing interest in the field has led to the release of multiple additional benchmarks and studies. Among them, Lai et al.'s [11] used their refined taxonomy of temporal anomaly types to generate 35 synthetic time series datasets, including 15 multivariate, and identify four real-world multivariate datasets with good anomaly type coverage. Their multivariate data generator however assumes individual dimensions to be independent, which is typically not the case in AIOps. Among their selected real-world datasets, the two that relate to our use case are "Web Attack" [28] and "Water Quality" [29]. The former corresponds to event-driven sequential data originally collected by the Canadian Institute for Cybersecurity in 2017, from which authors extract a single file containing 72 features derived from system metrics, 168,051 data records and 2,180 point-wise anomalies (from three kinds of intrusions). The latter corresponds to time series data collected by SPOTSeven Lab in 2018, extracted as a single file with 9 features, 139,566 data records and 51 pattern-wise anomalies. Both of these datasets therefore contain a single file primarily meant for online streaming anomaly detection, with no historical data assumed available for training, and anomalies currently not labeled according to the types defined by the authors.

In 2022, both Paparrizos et al. [30] and Schmidl et al. [19] released extensive evaluations and studies of time series anomaly detection methods, with efforts to lay out some new research insights. The former only focused on univariate methods, while the latter included both univariate and multivariate methods. Schmidl et al.'s study evaluated 71 algorithms from different domains on 24 data collections, including 12 real-world multivariate ones. The authors also developed "Good Time Series Anomaly Generator" (GutenTAG), to generate synthetic time series with a variety of labeled anomalies, included as part of their evaluation tool TimeEval [31].

Still in 2022, Alnegheimish et al. released the Signal Intelligence (Sintel) ecosystem, providing "systems and tools to design, develop and deploy AI applications on top of signals" [32]. This ecosystem includes the Orion library, which can be used to build and benchmark unsupervised time series anomaly detection methods, similarly to our Exathlon pipeline described in Section 3.3. While Orion defines its pipeline instances as sequences of preprocessing, modeling and postprocessing *primitives*, Exathlon defines them as sequences of configurable *scripts* for each step of an end-to-end explainable anomaly detection project. Both Orion and Exathlon's pipelines offer flexibility in terms of the preprocessing and anomaly detection methods used. Overall, these tools can be useful in different scenarios, with Exathlon providing the additional steps of data loading and data partitioning, but also *explanation discovery* for the detected anomalies, as well as a native AIOps integration through its dataset, metrics and methods.

Concurrently to these benchmarking efforts, a wide variety of multivariate time series datasets has been released and used in AIOps-related projects. According to a recent survey [9], the five most used as of 2023 were "Server Machine Dataset" (SMD) [33], "Soil Moisture Active Passive" (SMAP) satellite and "Mars Science Laboratory" (MSL) rover [34], "Water Distribution" (WADI) [35], and "Secure Water Treatment" (SWaT) [36].

SMAP and MSL were both collected by the NASA Jet Propulsion Laboratory. SMAP corresponds to 55 multivariate time series (called *channels*), with each one monitoring a given telemetry feature of the SMAP satellite (e.g., power, radiation), along with a binary vector encoding the commands sent and received for the satellite modules at each timestamp. MSL has a similar format, with 27 channels for the Mars rover. For both spacecrafts, channels are meant to be modeled separately (and thus considered independent). Overall, the datasets contain 496,444 telemetry values, with 105 anomaly ranges, labeled as either point-wise or contextual/collective anomalies.

SWaT and WADI were both collected by the iTrust Centre for Research in Cyber Security of the Singapore University of Technology and Design (SUTD), in collaboration with Singapore's water utility company. Both datasets consist of a single multivariate time series. In SWaT, this time series originates from the activity of a scaled-down but realistic water treatment plant. It consists of 51 features recorded every second for 11 days, from both the software and physical components (i.e., sensors and actuators) of the plant. A total of 36 attacks of various durations were conducted on the plant's components over the last four days of data. The WADI testbed extends the SWaT water plant by incorporating it into a complete water distribution system. It consists of 123 sensor and actuator values recorded every second for 18 days, with 15 attacks performed during the last two days.

SMD's use case may be the most similar to our work's, along with the one of "Application Server Dataset" (ASD) [13], released shortly after. Both datasets were collected by large Internet companies, as a collection of multivariate time series corresponding to independent machine or software *entities*. SMD provides 28 such entities, with 38 metrics (e.g., CPU load, network usage, memory usage) recorded every minute over five weeks. Every entity time series is meant to be modeled separately, with its second half labeled with anomalies of various lengths. ASD contains 12 entities, with 19 metrics (e.g., CPU, memory, network or VM-related) recorded every five minutes over 45 days, with the last 15 days labeled

Dataset	Sequences	Entities	Records	Features	Anomalies	R vs. P
Web Attack [28]	1	1	≈ 168k	72	**2,180**	Point
Water Quality [29]	1	1	≈ 140k	9	51	Range
SMAP [34]	55	**55**	≈ 430k	2	69	Both
MSL [34]	27	27	≈ 67k	2	36	Both
SWaT [36]	1	1	≈ 947k	51	36	Range
WADI [35]	1	1	≈ 1.2M	123	15	Range
SMD [33]	28	28	≈ 1.4M	38	327	Range
ASD [13]	12	12	≈ 154k	19	76	Range
Exathlon's [17]	**93**	10	**≈ 2.3M**	**2,283**	97	Range

Table 2.1: Total number of sequences, entities, records, features and anomalies for the most relevant existing real-world datasets and Exathlon's (with maximum values shown in **bold**), and whether they include *range-based* anomalies (covering multiple records), *point-based* anomalies (covering individual records), or both.

with anomalies of various lengths. Contrary to SMD, no entity in ASD experienced service changes during its recording period, which facilitates modeling under the assumption of no concept drift of normal data. Also, anomalies in ASD were further labeled as either *temporal*, *intermetric* or *intermetric-temporal*, as defined in [13]. Both SMD and ASD may contain a small amount of unlabeled anomalies in their training data.

Table 2.1 summarizes key statistics of these real-world multivariate datasets, and compares them to Exathlon's Spark Streaming dataset we propose and detail in Section 3.1. From this table, we can see that all existing datasets operate on a significantly smaller scale than Exathlon's in total number of sequences, data records and features, with Exathlon's also providing the second-largest number of range-based anomalies. Furthermore, our Spark Streaming dataset is the only one to cover the practical AIOps aspect of having **multiple sequences per entity** (i.e., Spark Streaming application here), recording each entity using the same features across multiple operation *contexts*, as we will detail in Chapters 4 and 5. With Exathlon's pipeline being modular and extensible, all of the existing datasets described can easily be incorporated into our benchmark.

2.1.2 Anomaly Explanation

Our release of the Exathlon benchmark took part in a recent stream of efforts to better evaluate and benchmark explainability methods [37, 38, 39, 40, 41, 42]. In the context of anomaly detection in multivariate time series, such methods typically seek to provide human-readable, meaningful explanations for the detection method's prediction function, mapping an input sample to its corresponding binary prediction or anomaly score.

Doshi-Velez et al. [38] categorized the evaluation of explainability into *application-grounded*, *human-grounded* and *functionally-grounded*. Both application-grounded and human-grounded evaluation require human evaluators. In application-grounded evaluation, those humans are domain experts, evaluating explanations with respect to a final application-specific task (e.g., doctors diagnosing diseases). In human-grounded evaluation, evaluators can be lay people, evaluating explanations with respect to a simplified task, typically used as a proxy for the target application (e.g., deciding the "best" out of two presented explanations). A practical example of application-grounded evaluation was FICO's "explainable machine learning" challenge held in 2018 [42]. In this challenge, contestants were asked to explain the predictions of a model deciding whether a customer should be granted a loan or not. The quality of these explanations were then evaluated by data scientists at FICO, partly relying on their domain knowledge.

Like most other recent works [37, 38, 39, 40, 41], our Exathlon benchmark focuses on *functionally-grounded* evaluation, where humans are kept out of the loop and quantitative metrics are used as proxies for interpretability. Defining such metrics requires making assumptions about the **type** (or **format**) **of explanation** expected. Nauta et al. [39] recently identified 14 categories of such types. In their study of explainable AI papers, they found that the format most commonly used when explaining individual predictions was **feature importance** scores, where each input feature is assigned a real-valued score reflecting its contribution to the prediction. This explanation format is natively supported by Exathlon, along with **feature localization** (i.e., binary feature importance), and **decision rules** (i.e., any interpretable logical formula using the input features and usable for classification).

Once an explanation format is defined, explanation *labels* may or may not be available for evaluation. Examples of datasets with explanation labels include SMD and ASD, where each test anomaly instance comes with "interpretation labels", defined as the dimensions that most contribute to the anomaly [33, 13]. Both papers assume feature importance scores as explanations, and evaluate them quantitatively with respect to feature localization labels. Overall, the metrics these papers introduce rely on the idea that 1) ground-truth important features for a given instance should be covered in its explanation, and 2) the k most important features of the explanation should be part of the ground-truth. Specifically, Su et al. [33] introduced the HitRate@P% metric, where P can be either 100 or 150, defined for a given instance as the proportion of the ground-truth feature set GT that is covered by the top-$\lfloor \text{P\%} \times |\text{GT}| \rfloor$ features of its explanation. Li et al. [13] further adapted this metric to a time series context through their InterPretation Score (IPS), where anomalous *ranges* (or *segments*) are considered instead of anomalous data points. For tabular data, Myrtakis et al. [41] framed the task of anomaly explanation as *feature subspace mining*, defining the expected explanation format as a ranked list of subspaces from the original feature space (i.e., ranked feature localization outputs), to be compared with one or more ground-truth subspaces for a given point anomaly. Explainers were then evaluated based on their ability to recover the ground-truth subspaces while also achieving high Precision for their top-ranked ones. Evaluation was conducted on both real-world datasets from the UCI machine learning repository [43] and synthetic datasets generated by Keller et al. [44].

In the absence of explanation labels, like in the Exathlon benchmark and numerous other scenarios [37, 39, 40], functionally-grounded evaluation can still be performed by designing **quantitative metrics** that reflect **desirable properties** to evaluate for an explanation. In 2023, Nauta et al. [39] identified 12 such properties, coined as *Co-12 properties*, and used this categorization scheme to analyze quantitative evaluation in explainable AI papers from 2014 to 2020. Among these properties, the four that most relate to the Exathlon benchmark so far are output-**Completeness**, **Continuity**, **Contrastivity** and **Compactness**.

Output-Completeness is a subcategory of Completeness. It measures the extent to which an explanation method agrees with the predictions of the model it explains. To quantify output-completeness, metrics can be derived from a *Deletion Check* or *Preservation Check*, which for feature importance scores would amount to removing or only keeping the important features returned by the explanation. When removing them (deletion check), the accuracy of the model should drop, while it should stay similar when removing every features except the important ones (preservation check). Other metrics can be derived from *Fidelity*, measuring how much the outcome produced by the explanation agrees with the outcome of the explained model. Exathlon covers this fidelity aspect for logical formulas through its "local accuracy" property with respect to binary model predictions.

Continuity reflects the fact that an explanation should remain stable for small variations

of the input that produce very similar predictions for the model. The main associated aspect is *Stability for Slight Variations*, measuring the similarity or discrepancy between explanations of an original input and some slightly different versions. For feature localization explanations, Exathlon captures this property through its "local instability" metric.

Contrastivity reflects whether explanations are "discriminative" with respect to ground-truth labels or other targets. An important aspect of this property is therefore *Target Sensitivity*, capturing "the intuition that class-specific features highlighted by an explanation should differ between classes", as stated by Pope et al. [45] in 2019. In our context of explainable anomaly detection, this can amount to requiring explanations of different anomaly *classes* be different.

Finally, Compactness reflects the intuitive requirement that an explanation should be sparse, short and/or non-redundant to be understandable by humans [39]. Practical metrics for this property include the *Size* of the explanation, measured differently depending on its format, as well as its *Redundancy*, measuring the information overlap within explanations for the formats that support it. In Exathlon, Size is measured as the number of features involved in an explanation.

2.2 Time Series Anomaly Detection in AIOps

This section reviews the time series anomaly detection methods that have been proposed for our AIOps use case. A first way to categorize such methods is based on whether they are built for *univariate* or *multivariate* data. In an AIOps context, univariate methods typically aim to detect abnormal behaviors in a single metric (e.g., Key Performance Indicator (KPI)) at a time. In contrast, multivariate methods aim to detect abnormal behaviors in a set of inter-dependent metrics that characterize a whole *entity* (e.g., a physical machine or running service) [9].

This work focuses on anomaly detection and explanation at the **entity level** (and thus on **multivariate** methods). This is indeed usually more *intuitive*, *effective* and *efficient*, due to four main reasons summarized by Su et al. [33]:

- "[I]n practice, operation engineers are more concerned about the overall status of an entity than each constituent metric".
- "[I]t is labor-intensive to train and maintain an individual anomaly detection model for each metric, given a large number of metrics".
- "[A]n incident (e.g., overload) at an entity typically causes anomalies in multiple metrics".
- "[I]ntuitively, modeling the expected value of one univariate time series can benefit from the more information in the multivariate time series of the same entity".

Multivariate time series anomaly detection methods have been categorized into *supervised*, *semi-supervised* and *unsupervised* methods [18, 46], considering full, partial or no label information for training, respectively. Due to the domain knowledge of IT operations required to reliably label anomalies, and the labor-intensive process of examining large amounts of time series data, collecting sufficient labels to employ supervised methods is often prohibitively costly [18, 16, 15]. As such, most of the anomaly detection literature, as well as this work, focuses on semi-supervised and unsupervised methods only. Specifically, Chapters 4 and 5 consider anomaly detection methods in an unsupervised setting, assuming unlabeled, mostly-normal data for training [18], and simply refer to unsupervised and semi-supervised methods assuming labeled *normal* data as **unsupervised**. Chapter 6 considers an extension of this unsupervised setting, assuming a few *anomaly* labels are available for training. In the literature, this updated setting is sometimes considered as a

subfield of *weakly-supervised* anomaly detection called "anomaly detection with incomplete supervision" [16, 15], which we simply refer to as **weakly-supervised** in this work.

2.2.1 Unsupervised Anomaly Detection in Multivariate Time Series

Numerous unsupervised anomaly detection methods in multivariate time series have been proposed throughout the years [8, 47, 48]. Schmidl et al. [19] recently introduced a taxonomy based on the way these methods derive their *anomaly scores* for data samples (the higher the score of a sample, the more it is deemed anomalous by the method). The only category we do not consider in this work is distance methods, which typically do not scale well with the large data dimensionality of our AIOps setting. In the following, we summarize each category and mention some of their relevant representatives (we refer the reader to the study of Schmidl et al. for a more comprehensive list).

Forecasting methods define anomaly scores of data samples as forecasting errors, that is, based on the distance between the forecast and actual value(s) of one or multiple data point(s) in a context window of length L. Popular methods in this category include LSTM-AD [49] and Graph Deviation Network (GDN) [50]. The former trains a stacked LSTM network to predict data points l time steps ahead. It then fits a multivariate Gaussian distribution to error vectors made by the model on a validation set, and defines the anomaly score of a test point as the negative likelihood of its error vector with respect to this distribution. The latter uses a graph attention-based forecasting network to predict the future behavior of its input time series from a graph where relationships between pairs of features are encoded as edges. It then uses this forecasting network to identify deviations and derive anomaly scores.

Reconstruction methods score data samples based on their reconstruction errors from a transformed space. Representative methods from this category include Principal Component Analysis (PCA) [51], Autoencoder (AE) [52, 53] and TranAD [14]. Both PCA and AE simply define anomaly scores of test vectors as their mean squared reconstruction errors from a transformed space. The transformation of PCA is a projection on the linear hyperplane formed by the principal components of the data. The one of AE is a non-linear mapping to a latent representation learned by a deep neural network trained to reconstruct input data from it. TranAD uses a transformer-based model with self-conditioning to gain training stability, an adversarial training procedure to amplify reconstruction errors and model-agnostic meta learning (MAML) to improve data efficiency. It then defines anomaly scores as averages of two reconstruction terms: one coming from a first decoder, and one coming from a second decoder whose input was augmented with the reconstruction loss of the first decoder as a focus score.

Encoding methods score data samples based on their deviation within a transformed space. Representatives of this category include Deep SVDD [54] and DCDetector [55]. Deep SVDD trains a deep neural network to map the input data to a latent representation enclosed in a small hypersphere, and then defines anomaly scores of test samples as their squared distance from this hypershere's centroid. DCDetector uses a dual-view attention structure based on contrastive learning to derive representations where differences between normal points and anomalies are amplified. It subdivides windows into adjacent "patches", with one view modeling relationships within patches and the other across patches. To derive anomaly scores, it then uses the insight that normal points tend to be similarly correlated for both views, while anomalies tend to be more correlated to their adjacent points than to the rest of the window.

Distribution methods define anomaly scores of data samples as their deviation from an estimated distribution of the data. The Mahalanobis method [56, 51] and Variational Autoencoder (VAE) [57] are representatives of distribution methods. The Mahalanobis method works by estimating the training data as a multivariate Gaussian distribution,

and defining the anomaly score of a test vector as its squared Mahalanobis distance from it (proportional to its negative log-likelihood with respect to the multivariate Gaussian). VAE trains a variational autoencoder network to estimate the distribution of the training data. The anomaly score of a given test point is then derived by drawing multiple samples from the probabilistic encoder, and averaging the negative log-likelihood of the reconstructions obtained from each of these samples.

Finally, **isolation tree methods** score data samples based on their "isolation level" from the rest of the data. Isolation forest [58] is the most popular isolation tree method. It relies on the assumption that anomalous samples are "few" and "different", which makes them quicker to isolate on average with random consecutive splits on attribute values than normal samples. As such, the method trains an ensemble of trees to isolate the samples in the training data, and defines the anomaly score of a test instance as inversely proportional to the average path length required to reach it using the trees.

Multiple efforts have also been made to propose **hybrid methods**, defining anomaly scores of data samples as combinations of scores coming from several scoring strategies above. Examples of hybrid methods include Multivariate Time-series Anomaly Detection via Graph Attention Network (MTAD-GAT) [59] and Auto-Encoder with Regression (AER) [60]. MTAD-GAT is the most representative in the literature, jointly optimizing a *forecasting* model and a *reconstruction* model on top of graph attention and GRU layers, and defining anomaly scores as convex combinations of reconstruction and forecasting errors. AER uses a bidirectional LSTM network to output the *reconstruction* of a sample along with the *prediction* of its first and last data records, and defines anomaly scores as either convex combinations or weighted point-wise products of reconstruction and prediction errors. Although hybrid methods could be considered in extensions of our studies, we choose not to include them in this work to simplify our analyses across the scoring strategy dimension. Additionally, the most representative hybrid method, MTAD-GAT, has been shown to be outperformed by the more-recent TranAD method [14].

2.2.2 Weakly-Supervised Anomaly Detection and Contrastive Learning

Despite the success of unsupervised methods, the total absence of anomaly labels to rely on can prevent them from automatically differentiating unusual patterns that constitute *relevant* anomalies for the analysts from less interesting "*noise*" that just hinders the analysis [61]. An efficient solution to this problem can be to gather labels for a few relevant anomalies, and incorporate this label information into anomaly detection models. This setting corresponds to the growing field of *weakly-supervised* anomaly detection [16, 15], where a vast majority of unlabeled, assumed mostly-normal, data gets augmented with limited anomaly labels.

In the literature, weakly-supervised methods relevant to our use case have been categorized into *anomaly feature representation learning*, *anomaly score learning*, *active learning* and *reinforcement learning* methods [15]. The methods we propose in Chapter 6, Contrastive and Triplet Encoder for Anomaly Detection with a Few Anomaly Labels (CEADAL and TEADAL, respectively), belong to the **anomaly feature representation learning** category, aiming to learn a transformation to a latent space where normal and anomalous data samples are clearly separated. This category, along with anomaly score learning, has the benefit of not requiring any explicit request to the user.

Our work compares the performance of CEADAL and TEADAL to another anomaly feature representation learning method called Deep SAD [62], a follow-up work to Deep SVDD [54] that improves on its latent mapping by imposing that anomalous samples get sent far from its normal hypersphere's centroid. Deep SAD has a similar spirit to CEADAL, in its way of applying a strong proximity constraint for normal samples in latent space, which, as detailed in Chapter 6, will turn out to be very effective in our

AIOps setting. Like CEADAL, Deep SAD forces normal samples to belong to a tight latent region, with anomalies being mapped outside of it. For Deep SAD, this tight region is defined around a randomly initialized centroid, while CEADAL defines it indirectly through contrastive learning.

Both CEADAL and TEADAL rely on a **contrastive learning** framework [22, 23]. CEADAL relies on the contrastive loss originally proposed by Hadsell et al. [22], adapted here by defining samples as "similar" (or *concordant*) if they belong to the same class (normal or anomalous), and as "dissimilar" (or *discordant*) otherwise. TEADAL relies on the more recent and less constrained *triplet loss* [23], adapted here by setting normal samples as anchor and positive inputs, and anomalous samples as negative inputs.

An example of method that leveraged the contrastive loss for a similar purpose is "Contrastive Autoencoder for Drifting detection and Explanation" (CADE) [63]. This method trains an autoencoder network with a contrastive loss to map samples from each of multiple (balanced) classes to a distinct and tight cluster in latent space. It then defines drifting samples as those that get mapped unusually far away from every class cluster. In contrast, CEADAL considers only two classes, one normal and one anomalous, with the normal class assumed highly prevalent in the data. It then trains an encoder to cluster samples from the normal class only, away from anomalous data, but not imposing any proximity constraint within samples of the anomalous class. Finally, an important aspect of both CEADAL and TEADAL is to make use of the *online batching strategy* of [23] in constituting sample pairs and triplets, respectively, with the models being trained on the "hardest" examples only (i.e., the most "subtle" anomalies or "noisy" normal patterns in our anomaly detection context).

2.2.3 Domain Generalization

As we will see in Chapter 5, a useful framework to address the normal behavior shift challenge of our AIOps setting is **domain generalization** (DG). In this framework, data samples are collected from multiple, distinct *domains*, with certain characteristics of the observed data being determined by the domain, and others being independent from it. The goal is then to build models on a set of training (or *source*) domains that can generalize to another set of test (or *target*) domains.

Domain generalization has been mainly studied in the context of *image classification*, with the domains usually corresponding to the way images are represented or drawn. DG methods can broadly be categorized as based on *explicit feature alignment*, *domain-adversarial learning* or *feature disentanglement* [20, 21]. Explicit feature alignment methods seek to learn data representations where feature distribution divergence is explicitly minimized across domains, with divergence metrics including maximum mean discrepancy (MMD), statistical moments, second-order correlations, Wasserstein distance or Kullback-Leibler (KL) divergence [20, 21]. Rather than using such divergence metrics directly, domain-adversarial learning methods seek to minimize domain distribution discrepancy through a minimax two-player game, where the goal is to make the features confuse a domain discriminator [64], usually implemented as a multiclass or binary domain classifier [65, 66, 67, 68]. Such adversarial methods can suffer from instabilities that make them hard to reproduce [69, 70], while explicit feature alignment can become very costly when the number of source domains gets large, like in our AIOps setting. For these reasons, this work considers domain generalization based on feature disentanglement [71, 72, 73], where methods seek to decompose the input data into *domain-specific* and *domain-invariant* features, and perform their tasks in domain-invariant space.

Domain-Invariant VAE for Anomaly Detection (DIVAD), which we propose in Chapter 5, specifically relates to Domain-Invariant Variational Autoencoders (DIVA) [71], designed for image classification. DIVA uses variational autoencoders (VAEs) to decompose

input data into domain-specific, class-specific and residual latent factors, conditioning the distributions of its domain-specific and class-specific factors on the training domain and class, respectively, and enforcing this conditioning by using classification heads to predict the domain and class from the corresponding embeddings. It then uses its class-related classifier to derive its predictions for test images. Because of this class supervision, this method cannot be applied to our unsupervised anomaly detection setting, hence motivating the design of DIVAD.

Domain generalization for time series AD recently gained some attention through anomalous sound detection and the DCASE2022 Challenge, where the task was to identify whether a machine was normal or anomalous using only normal sound data under domain-shifted conditions [74]. The methods proposed as part of this challenge however modeled single-channel (univariate) sound waves, while also relying on sound-specific preprocessing (e.g., magnitude spectrograms), and assuming labels such as the machine state, the type of machine, domain shift or noise considered to train domain-invariant or disentangled representations [75]. This makes these methods setting-specific in nature, and thus unsuitable to solve our more general problem of unsupervised AD in AIOps.

2.3 Explanation Discovery

As we will see in Chapter 3, we constructed the Exathlon benchmark to assess the performance of both anomaly *detection* and *explanation* methods, with this latter step referred to as *Explanation Discovery* (ED). This section introduces and categorizes the main ED methods mentioned and targeted by our benchmark.

2.3.1 Interpretable Machine Learning

Interpretable machine learning has recently attracted a lot of attention [76], with relevant techniques being generally categorized into *model-specific* and *model-agnostic* methods. Model-specific methods include intrinsically interpretable models, which directly build a human-readable model from the data (e.g., linear or logistic regression, shallow decision trees or rules) [76], as well as methods that can only be applied for specific model characteristics (e.g., neural network feature visualization [76]). In contrast, model-agnostic methods separate explanations from the explained machine learning (ML) model, typically relying on input and output pairs only, which offers the flexibility to mix and match ML models with interpretation methods. In the model-agnostic family, several methods obtain interpretable classifiers by perturbing the inputs and observing the response [77, 78, 79, 80]. "Local Interpretable Model-agnostic Explanations" (LIME) [78] explains a prediction of any classifier by approximating it locally with an interpretable sparse linear model, and explains the overall model by selecting a set of representative instances with explanations. As such, it is generally considered a method producing *local explanations*. Anchors [79] improved upon LIME by replacing its linear model with a logical rule for explaining a data instance. It offers better coverage of data points in a local neighborhood, but does not provide support for time series data. SHAP scores [81], RESP scores [82], and axiomatic attribution [80] are also instance-level explanations that assign a numerical score to each feature, representing their importance in the outcome. In contrast to local explanations, other work aims to explain a model via *global explanations*. Some of them approximate a deep learning model using a decision tree [83, 84], or by learning a decision set [77, 85] directly as explainable models.

In the context of the Exathlon benchmark, we refer to interpretable machine learning methods broadly as **model explainers**. The goal of such ED methods is indeed to explain the prediction process of a particular model, which makes them rely (at least)

on this model's decision function, mapping an input sample to its corresponding binary prediction or anomaly score. Specifically, the Exathlon benchmark and pipeline currently support *model-agnostic* methods only, and were primarily designed to build and evaluate *local explanations* (although global explanations can also be considered via the ED2 criteria described in Section 3.2.2).

2.3.2 Outlier Explanation

Besides interpretable machine learning, several efforts have been made to explain anomalies (or *outliers*) given a set of normal and anomalous data records annotated by a user or a detection model [86]. These methods are supported by our Exathlon benchmark under the name of **data explainers**, since they aim to explain differences between anomalous and reference *datasets*, only relying on AD models to define those datasets through their prediction labels. Specifically, our benchmark currently supports the evaluation of data explainers that provide their explanations in the form of *feature importance scores* or *decision rules* involving the input features.

Representative methods defining outlier explanations as feature importance scores are LookOut [87] and Contextual Outlier INterpretation (COIN) [88]. LookOut is based on *subspace search* [86], deriving the explanation of an outlier as a set of *focus-plots*, where a focus-plot is defined as a 2-dimensional scatter plot of all data points projected on a pair of input features. It first computes the anomaly scores of the explained outlier for each possible focus-plot using any AD method (e.g., isolation forest [58]). It then searches for the subset of b focus-plots that best collectively explains the outlier (i.e., maximizes its sum of anomaly scores), using a greedy algorithm with approximation guarantees to address the NP-hard optimal subset selection problem. The retained focus-plots can then serve as visual (or *pictorial* [87]) explanations, or feature importance scores can be derived for the focus-plots' features, by summing the anomaly scores they produced for the outlier. COIN relies on a different strategy, deriving the explanation of an outlier o_i based on its *local neighborhood* [86] (or *context* [88]). It first identifies the context C_i of o_i, defined as its nearest normal neighbors based on the L2-norm. It further segments C_i into L disjoint clusters, $C_i = \{C_{i,1}, C_{i,2}, \cdots, C_{i,L}\}$, using K-means or hierarchical clustering and ignoring clusters of small sizes in subsequent steps. For each cluster l, COIN then fits a 1-norm SVM [89] with a linear kernel to separate normal points from the outlier, producing a weight vector $\boldsymbol{w}_{i,l}$. This weight vector is then used to derive the importance score of a feature a_m for cluster l, as $s_{i,l}(a_m) = |w_{i,l}[m]|/\gamma^m_{i,l}$, where $|w_{i,l}[m]|$ is the absolute value of the element at index m in $\boldsymbol{w}_{i,l}$, and $\gamma^m_{i,l}$ is the average distance along the mth axis between an instance in $C_{i,l}$ and its closest neighbors. The global importance score of a_m for o_i is finally defined as its average score across the clusters, weighted by the cluster sizes.

Representative methods defining outlier explanations as decision rules are EXstream [90] and MacroBase [91]. EXstream relies on *entropy-based rewards* [86] to explain a given abnormal interval I_A with respect to a reference interval I_R, both in the form of a decision rule and feature importance scores. It first computes single-feature rewards from an entropy-based distance function, measuring the *segmentation* of feature values across I_A and I_R (the less segmentation of values, the more important the feature). Given single-feature reward ranks, it then employs mechanisms to filter unimportant features, defining them as those (i) having reward ranks below a sharp drop in the ranked list of rewards (*reward leap filtering*), (ii) having only spurious correlations with I_A (*false positive filtering*), and (iii) having information overlap with more important features (filtering by *correlation clustering*). To form the explanation of I_A, EXstream then relies on the segmentation of values produced for each retained feature, using single-predicate rules for features with only one boundary (e.g., $f_1 \leq 10$, where f_1 refers to the first feature), or composed predicates in case of multiple abnormal value ranges (e.g., $f_2 < 20 \vee (f_2 \geq 30 \wedge f_2 \leq 50)$),

and further combines partial, feature-wise explanations with conjunctions. MacroBase derives its explanations based on *frequent itemset mining*, where items can be mapped to predicates in our context by discretizing continuous data (e.g., using equi-width partitioning, like in [92]). For an explained outlier set and corresponding set of normal points (or *inliers* [91]), MacroBase first searches individual items with minimum support in both outliers and inliers. It then defines the "importance" of an item (or itemset) as its relative *risk ratio*, quantifying how much more likely a data point is to be an outlier if it satisfies the rule of the item(set), as opposed to the general population [91, 93]. Given its goal of finding itemsets whose subsets each have minimum support and risk ratio, MacroBase accelerates computations by first computing risk ratios for single items, and then support of itemsets whose members have sufficient risk ratios. Once important itemsets have been retrieved, they can be combined in various ways to form the explanation of the outlier set (e.g., as its itemset of highest risk ratio).

3 The Exathlon Benchmark

This chapter presents **Exathlon** [17], the first public benchmark for explainable anomaly detection over high-dimensional time series, which we will rely on in the following chapters. Exathlon provides a benchmarking platform that consists of (i) a curated **dataset**, centered around a Spark Streaming application monitoring use case, (ii) a flexible **evaluation methodology** to assess the performance of anomaly detection and explanation methods, and (iii) an end-to-end explainable anomaly detection **pipeline** to facilitate the usage and configuration of the benchmark, as well as the implementation of new methods. We present each of these components in Sections 3.1, 3.2 and 3.3, respectively. The dataset, code and documentation of Exathlon are publicly available at `https://github.com/exathlonbenchmark/exathlon`.

To further diagnose the predictions of anomaly detection and explanation methods, we also released a visualization platform called **LEADS Viewer**. This platform provides a Graphical User Interface (GUI) to visualize explainable anomaly detection outputs on high-dimensional time series data, be them generated using Exathlon's pipeline or not. The code and documentation of LEADS Viewer are publicly available at `https://github.com/exathlonbenchmark/leads-viewer`.

3.1 Spark Streaming Dataset

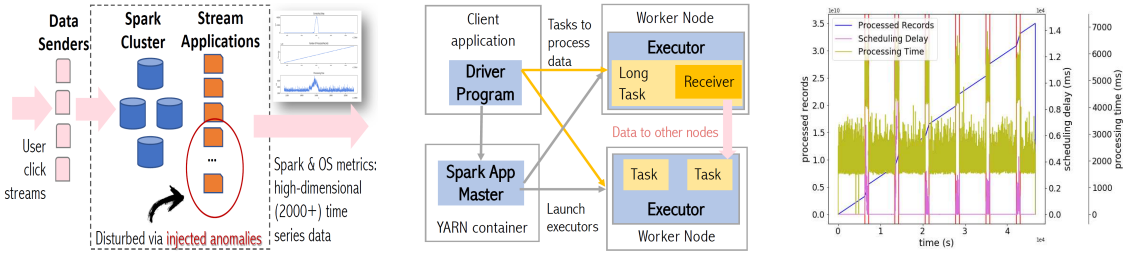
This section presents the Spark Streaming dataset of the Exathlon benchmark. Apache Spark is an open-source distributed processing engine that greatly facilitates executing data engineering, data science, or machine learning tasks on single-node machines or clusters. It is today the most popular engine for scalable computing, used by thousands of companies including 80% of the Fortune 500 [94]. Spark Streaming (now Structured Streaming) expands upon Spark to provide efficient data stream processing at scale. In the wake of core Apache Spark, Spark Streaming services are today extensively deployed to serve as central components of a lot of company workflows. Yet, such services are especially incident-prone by nature, as they need to operate with very low latency on large-volume streams that may come from various external sources.

This makes the task of detecting and explaining anomalies in Spark Streaming operations particularly relevant to a vast number of real-world scenarios, while also reflecting well practical AIOps challenges. For these reasons, we selected this particular task as the central use case of the Exathlon benchmark.

3.1.1 Data Collection

To collect the dataset of Exathlon, we considered a Spark workload of 10 stream processing **applications** (further described in Appendix A.1) analyzing user click streams from the WorldCup 1998 website [95].[1]. As in Figure 3.1a, data sender servers send streams at a controlled **input rate** to a Spark cluster of four nodes. Each application has certain workload characteristics (e.g., CPU or I/O intensive) and is executed by Spark in a distributed

[1] The collection of the Spark Streaming dataset was performed prior to my arrival in the project, mainly by Arnaud Stiegler [96]. We later re-organized, formalized and documented this dataset into the version integrated in Exathlon, with some traces and event types renamed, as well as the added notion of "extended anomaly intervals" presented below.



(a) Spark application monitoring.

(b) Spark execution environment.

(c) Trace with bursty input anomalies.

Figure 3.1: Spark application monitoring and metrics observed in anomaly instances (a pair of red vertical bars marks a root cause event).

Metric Type	Spark UI Driver	Spark UI Executor	OS (`nmon`)
# of Metrics	243	5 x 140 = 700	4 x 335 = 1340
Total	2,283		
Frequency	1 data item per second		
Data Items	2,335,781		
Duration	649 hours		
Total Size	24.6 GB		

Table 3.1: Collected metrics and data size.

manner, as in Figure 3.1b. Submitted an application, Spark launches a *driver* process to coordinate the execution. The driver connects to a resource manager (Apache Hadoop YARN), which launches *executor* processes on a subset of nodes where tasks (units of work on a data partition, e.g., `map` or `reduce`) will be executed in parallel [17].

Given 32 cores, each node can also run tasks from multiple applications concurrently. As common real-world practice, we run 5/10 randomly selected applications at a time. The placement of driver and executor processes to cluster nodes is decided by YARN based on data locality and load on nodes, among other factors. Except for I/O activities, YARN offers container isolation for resource usage of all parallel processes.

We ran the 10 Spark Streaming applications in our four-node cluster over a 2.5-month period. We refer to the data collected from each run of an application as a **trace**. Some of the collected traces were removed because they were affected by cluster downtimes or the injected anomalies were not well reflected in the data due to failed attempts. After this manual pruning, we retained 93 traces to constitute the Spark Streaming dataset of Exathlon.

Collected Metrics

Table 3.1 gives a summary of the metrics collected per trace. Each given trace was stored as a CSV file recording the evolution of both Spark User Interface (UI) and Operating System (OS) metrics every second. Spark UI metrics relate to the Spark processes of the recorded application (e.g., scheduling delay, statistics on the streaming data received and processed), available through the Spark Monitoring and Instrumentation interface [97], while OS metrics relate to the operating system of the four cluster nodes, collected via the `nmon` software [98].

The driver offers 243 metrics (e.g., scheduling delay, statistics on the streaming data received and processed). Each executor provides 140 metrics (e.g., CPU time and runtime, heap memory used). As we wanted to keep the number of metrics the same for all traces,

Trace Type	Anomaly Type	# of Traces	Anomaly Instances	Anomaly Length (RCI + EEI) min, avg, max	Data Items
Undisturbed	N/A	59	N/A	N/A	1.4M
Disturbed	T1: Bursty input	6	29	15m, 22m, 33m	360K
Disturbed	T2: Bursty input until crash	7	7	8m, 35m, 1.5h	31K
Disturbed	T3: Stalled input	4	16	14m, 16m, 16m	187K
Disturbed	T4: CPU contention	6	26	8m, 15m, 27m	181K
Disturbed	T5: Driver failure	11	9	1m, 1m, 1m	128K
Disturbed	T6: Executor failure		10	2m, 23m, 2.8h	
Ground Truth	(`app_id`, `trace_id`, `anomaly_type`, `root_cause_start`, `root_cause_end`, `extended_effect_start`, `extended_effect_end`)				

Table 3.2: Undisturbed traces, disturbed traces and ground-truth labels for 97 anomalies.

we set a fixed limit of five for the number of Spark executors (two or three active + three or two backup). This way, even if an active executor failed during a run and a backup took over, the number of metrics collected stayed the same, $5 \times 140 = 700$, with null values set for inactive executors. 335 OS metrics were collected for each of the four cluster nodes, including CPU time, network traffic and memory usage. All in all, each trace consists of a total of 2,283 metrics recorded each second for 7 hours on average, constituting a multi-dimensional time series.

Undisturbed and Disturbed Traces

In an approach similar to chaos engineering (i.e., injecting failures into a production system to verify/improve its reliability) [99] and general systems monitoring (e.g., Microsoft's NetMedic [100]), we first collected **undisturbed** traces to characterize the normal behavior of our Spark applications and cluster, and then introduced various anomalous events to collect **disturbed** traces. Table 3.2 provides an overview.

Undisturbed Traces Uninterrupted executions of 5/10 randomly selected applications at a time, at parameter settings within the capacity limits of our Spark cluster, over a period of one month, gave us 59 undisturbed traces of 15.3GB in size. Any instances of occasional cluster downtime were manually removed from these traces. It is important to note that, although undisturbed, these traces still exhibit occasional variations in metrics due to Spark's inherent system mechanisms (e.g., checkpointing or CPU usage by a DataNode in the distributed file system). Since such variations do appear in almost every trace, we consider them as part of the normal system behavior. In other words, our normal data traces include some "noise", as most real-world datasets typically do.

Disturbed Traces Disturbed traces were obtained by introducing anomalous events during application executions. In total, we collected 34 disturbed traces (9.3 GB in size) covering six types of injected events, designed such that they:

1. Led to visible effects in traces.
2. Did not lead to an instant crash of the application (since anomaly detection would be of little help in this case).
3. Could be tracked back to their root causes.

The event types are the following (please refer to Appendix B.2 of our Exathlon paper [101] for further details):

- **Bursty Input (Type 1).** To mimic input rate spikes, we run a disruptive event generator (DEG) on data senders to temporarily increase the input rate by a given

factor for a duration of 15-30 minutes. We repeat this pattern multiple times during a given trace collection, creating a total of 29 instances of this event type over six different traces. Figure 3.1c shows an example of bursty input anomalies.

- **Bursty Input Until Crash (Type 2).** This is a longer version of Type 1 anomalies, where we let the DEG period last forever, crashing the executors due to lack of memory. When an executor crashes, Spark launches a replacement, but the sustained high rates keeps crashing the executors, until Spark eventually decides to kill the whole application. We injected this anomaly into seven different traces.
- **Stalled Input (Type 3).** This type of event mimics failures of Spark data sources (e.g., Kafka or HDFS). To create it, we run a DEG that set the input rates to zero for about 15 minutes, and then periodically repeat this pattern every few hours, giving us a total of 16 anomaly instances across four different traces.
- **CPU Contention (Type 4).** The YARN resource manager cannot prevent external programs from using the CPU cores that it has allocated to Spark processes, causing scheduling delays to build up due to CPU contention. We reproduced this anomaly using a DEG that ran Python programs to consume all CPU cores on a given Spark node. We created 26 such anomaly instances over six different traces.
- **Driver Failure (Type 5) and Executor Failure (Type 6).** Hardware faults or maintenance operations may cause a node to fail all of a sudden, making all processes (drivers and/or executors) located on that node unreachable. Such processes must be restarted on another node, which causes delays. We created such anomalies by failing driver processes, where the number of processed records drops to zero until the driver comes back up again in about one minute (with all counter metrics reset). We also created anomalies by failing a given executor process. When an executor fails, it may or may not get replaced with another one. In all cases, we observe a buildup in the application's delays while operating with one fewer executor, which either ends with the executor replacement or an application crash if the executor is not replaced. We created nine driver failures and 10 executor failures over 11 different traces.

Ground-Truth Table For all of the 97 anomaly instances we injected over 34 disturbed traces, we provide ground truth labels with the information shown in Table 3.2. Such labels include both *root cause intervals* (RCIs) and their respective *extended effect intervals* (EEIs). RCIs typically correspond to the time period during which DEG programs were running, whereas the EEIs are the time periods that start immediately after an RCI and end when important system metrics return to normal values or the application is eventually pushed to crash. The EEIs are manually determined using domain knowledge. Additional details can be found in Appendix A.2.

3.2 Evaluation Methodology

This section presents the evaluation methodology we proposed to benchmark Anomaly Detection (AD) and Explanation Discovery (ED) methods. As summarized in Table 3.3, Exathlon was designed to evaluate AD and ED algorithms in both **functionality** and **computational performance**, using well-defined metrics, while also allowing users to specify their own through the pipeline presented in Section 3.3.

3.2.1 Anomaly Detection (AD) Functionality

To best fit its Spark Streaming dataset, Exathlon was primarily designed to target *range-based anomalies*, occurring over contiguous time intervals instead of single time points.

	Anomaly Detection (AD) Functionality	Explanation Discovery (ED) Functionality	Computational Performance
Evaluation Criteria	AD1: Anomaly Existence AD2: Range Detection AD3: Early Detection AD4: Exactly-Once Detection	ED1: Local Explanation ED2: Global Explanation	P1: AD Training Scalability P2: AD Inference Efficiency P3: ED Efficiency
Evaluation Metrics	Accuracy: *Range-based Precision, Recall, F-Score, AUPRC*	Conciseness: *Size* Consistency: *Instability* (ED1), *Discordance* (ED2) Accuracy: *Point-based Precision, Recall, F-Score*	Time, given different *Dimensionality* and *Cardinality* factors

Table 3.3: The Exathlon evaluation methodology and benchmark design.

Support for *point-based* evaluation was however also added, and used in Chapter 4.

Evaluation Criteria

We identified four key criteria for evaluating AD functionality, listed below from basic toward advanced, where a higher AD level includes the requirements of all preceding levels:

- **AD1 (Anomaly Existence).** The first expectation is to flag the existence of an anomaly somewhere within the *anomaly interval* (i.e., the root cause interval (RCI) + the extended effect interval (EEI))[2].
- **AD2 (Range Detection).** The next expectation is to report not only the existence, but also the precise time range of an anomaly. The wider a range of an anomaly that an AD method can detect, the better its understanding of the underlying real-world phenomena.
- **AD3 (Early Detection).** The third expectation is to minimize the *detection latency*, i.e., the difference between the time an anomaly is first flagged and the start time of the corresponding RCI.
- **AD4 (Exactly-Once Detection).** The last expectation is to report each anomaly instance exactly once. Duplicate detections are undesirable, because they may not only redundantly cause repeated alerts for a single anomalous event, but also trouble in understanding whether those alerts relate to the same anomaly event or not.

Evaluation Metrics

To assess how well an AD method can meet these four functionality levels, we use the framework of Precision and Recall for time series [106], proposed in 2018 to evaluate *real* and *predicted* anomalies at the *range* level with a set of tunable parameters. By setting the values of these parameters in a particular way and applying the resulting Precision/Recall formulas to the output of an AD algorithm, one can assess how well that output measures up to the quality expectations represented by those parameter settings. We leverage this as a mathematical tool to quantify how well an AD algorithm meets AD1-AD4. Furthermore, we chose to do this in a way that every level AD$\{i\}$ builds on and adds to the requirements of the previous level AD$\{i-1\}$. This monotonic design ensures that the AD functionality score that an algorithm gets (Precision, Recall, or other metrics derived from them) is always ordered as:

$$\text{score(AD1)} \geq \text{score(AD2)} \geq \text{score(AD3)} \geq \text{score(AD4)},$$

[2]Note that this criterion corresponds to the popular "point adjustment" protocol used in numerous works [55, 102, 103, 33, 104], but also heavily criticized nowadays as greatly over-estimating performance, being unpredictable and misleading [105]. Our advice would therefore be to either avoid this criterion or always use it together with other ones and/or visual inspections.

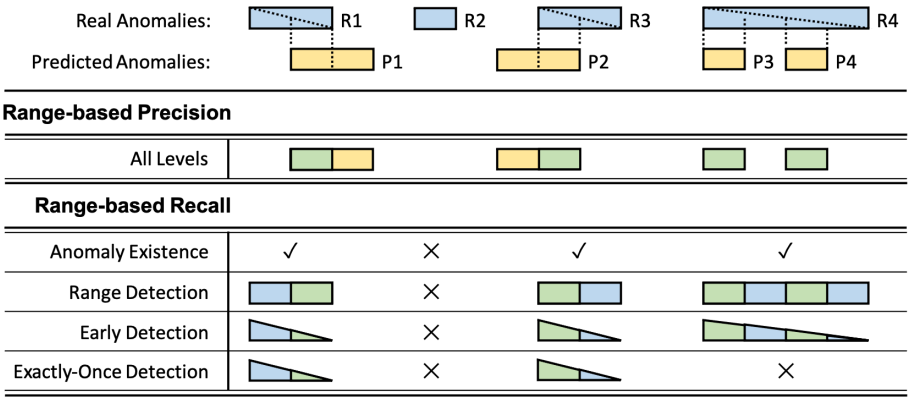


Figure 3.2: Range-based Precision and Recall at AD levels 1-4. Precision evaluates prediction quality (green out of green + yellow for each Pi). Recall evaluates anomaly coverage (green out of green + blue for each Ri).

which facilitates evaluating and interpreting results in a systematic way. In Exathlon, we preferred this design over the alternative of treating each AD level as orthogonal to enable users to develop and perfect their models for tasks that become increasingly challenging.

Figure 3.2 provides a simple example to illustrate how range-based Precision and Recall are computed for different AD levels. Given real anomaly ranges R1 . . R4 and predicted anomaly ranges P1 . . P4 produced by an AD algorithm, we first compute Precision and Recall for each range and then average them for overall Precision and Recall. Intuitively, Precision focuses on the size of True Positive (TP) ranges (colored green) relative to TP + False Positive (FP) ranges (colored yellow), while Recall focuses on the size of TP ranges relative to TP + False Negative (FN) ranges (colored blue). For AD1, Recall(Ri) is 1 if Ri is flagged, 0 otherwise. For AD2, Recall(Ri) is proportional to the relative size of the TP range. For AD3, Recall(Ri) is further weighted by position of the TP range relative to the start of Ri. Finally, at AD4, Recall(Ri) degrades to 0 for any Ri that is not flagged using exactly one predicted range. Precision(Pi) is computed in an analogous way, except that AD levels about anomaly coverage quality (AD1 and AD3) are not relevant to it; rather, the main focus is on the size and number of the real anomaly ranges that are successfully predicted. In our simple example, it turns out that all AD levels for Precision consider the same Pi subranges.

Formally, we recall that, given sets of real anomaly ranges $R = \{R_1, R_2, \ldots, R_{N_r}\}$ and predicted anomaly ranges $P = \{P_1, P_2, \ldots, P_{N_p}\}$, the *range-based Recall* of a *real* anomaly range R_i is defined as:

$$\text{Recall}_T(R_i, P) = \alpha \cdot \text{ExistenceReward}(R_i, P) + (1 - \alpha) \cdot \text{OverlapReward}(R_i, P),$$

with $\alpha \in [0, 1]$ a tradeoff parameter balancing the importance of rewarding the detection of R_i's *existence* and the *overlap* between R_i and P. The existence reward is defined as:

$$\text{ExistenceReward}(R_i, P) = \begin{cases} 1 & \text{if } \sum_{j=1}^{N_p} |R_i \cap P_j| \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

The overlap reward captures the *size*, *position* and *cardinality* aspects of the overlap between R_i and P through parameter functions ω, δ and γ, respectively. It is defined as:

$$\text{OverlapReward}(R_i, P) = \text{CardinalityFactor}(R_i, P) \cdot \sum_{j=1}^{N_p} \omega(R_i, R_i \cap P_j, \delta).$$

	Precision Parameters			Recall Parameters		
AD Functionality Level	α	δ	γ	α	δ	γ
AD1: Anomaly Existence	0	Flat	1	1	N/A	N/A
AD2: Range Detection	0	Flat	1	0	Flat	1
AD3: Early Detection	0	Flat	1	0	Front	1
AD4: Exactly-Once Detection	0	Flat	0	0	Front	0

Table 3.4: Range-based Precision/Recall parameter settings for our AD1-4 levels.

The cardinality term therefore serves as a scaling factor for the rewards earned from the size and position of R_i and P overlaps. It is maximized when R_i overlaps with *at most one* predicted range from P, and is defined through the γ function otherwise:

$$\text{CardinalityFactor}(R_i, P) = \begin{cases} 1 & \text{if } R_i \text{ overlaps with at most one } P_j \in P, \\ \gamma(R_i, P) & \text{otherwise.} \end{cases}$$

The *range-based Precision* of a *predicted* range P_i with respect to the set of real anomaly ranges R is defined similarly, except it has no existence reward component (i.e., $\alpha = 0$):

$$\text{Precision}_T(R, P_i) = \text{CardinalityFactor}(P_i, R) \cdot \sum_{j=1}^{N_r} \omega(P_i, P_i \cap R_j, \delta).$$

Table 3.4 summarizes the parameter values we set for range-based Precision and Recall to reflect our different AD levels. We provide a brief description of these parameters here, and refer the reader to the original paper for further details [106]. The α parameter is only relevant here in the context of Recall, and when (only) focusing on anomaly existence. We therefore set it to 1 when computing AD1's Recall, and always to 0 otherwise. The δ parameter is a function defining the reward of a detected (or real) anomaly range based on its overlap *position*. We set this parameter to "Front" (i.e., linearly decreasing rewards over time) for AD3-4 Recall, to penalize detection latency for the predictions in real anomaly ranges. For other scenarios, we set this parameter to "Flat" (constant rewards over time), to reflect that overlap position is not important. Finally, our AD1-3 levels do not penalize one-to-many mappings for either real or predicted anomaly ranges, which corresponds to setting the γ function to the constant, maximal value of 1 for both Precision and Recall. Our AD4 level, however, fully cancels the rewards of both (i) repeated predictions in real anomaly ranges, and (ii) predictions spanning multiple real ranges. This level indeed requires exact, one-to-one mappings between real and predicted ranges, which corresponds to setting γ to the constant, minimal value of 0 for both Precision and Recall.

Like reminded above, the range-based Precision/Recall framework provides a fourth parameter ω, which serves to compute how much reward is earned from the size of the overlap between a real and predicted anomaly range (relevant for AD2 and beyond). We use its default, additive definition from the original paper here, with a minor normalization adjustment to ensure monotonicity. More specifically, we make the Precision/Recall scores decrease from AD2 to AD3 by defining the union of detected ranges under AD2 as the best-positioned (i.e., earliest) detected range under AD3 to achieve a monotonic behavior.

Although both the AD methods and metrics considered by Exathlon focus on *binary* anomaly detection (normal vs. anomalous ranges), our dataset is inherently *multiclass* (normal ranges vs. six types of anomalous ranges). This raises a question on how to evaluate binary predictions under multiclass labels. Exathlon takes a holistic approach, and allows to consider AD predictions evaluation (i) globally, (ii) grouped by event class/type

and (iii) averaged across types. Even though a binary predictor is not able to detect different event types, we can indeed still measure its resulting coverage (i.e., Recall) for each type. Type-wise measurement is however not entirely meaningful for Precision, since false positives are essentially typeless.

3.2.2 Explanation Discovery (ED) Functionality

Once an anomalous instance was flagged by a detection method, the next desirable functionality is to find the best explanation for the anomaly detected, or more precisely, a human-readable formula offering useful information about what has led to the anomaly.

Many ED methods have been proposed in recent years. These methods differ in the form of "explanation" provided, with some returning a logical formula as an explanation (e.g., EXstream [90], MacroBase [91] or Anchors [79]), others returning a decision tree (like Tree Regularization [84]), and some others returning a numerical score for each feature, such as coefficients of a linear model (like LIME [78]) or SHAP scores [81]. Exathlon does not pose heavy restrictions on the form of explanation used, but rather takes an abstract view of explanations. Formally, we model each trace in a test dataset as a multivariate time series (or *sequence*):

$$\boldsymbol{S} = [\boldsymbol{s}_1 \cdots \boldsymbol{s}_t \cdots \boldsymbol{s}_T]^T,$$

where each $\boldsymbol{s}_t \in \mathbb{R}^M$ is a data vector, or *record*, with M features. We define the data from a *predicted anomaly range* starting at index $t \in [1 \,..\, T]$ and spanning $l > 0$ records in this sequence as:

$$P_{t,l} := \boldsymbol{S}_{t:t+l-1} = [\boldsymbol{s}_t \cdots \boldsymbol{s}_{t+l-1}]^T.$$

We then denote the *explanation* generated for this predicted anomaly as $E_{t,l}$, and treat it as a function of the features $\mathcal{A} = (a_1, \ldots, a_M)$ from the data[3]:

$$E_{t,l}(a_1, \ldots, a_M) \models P_{t,l},$$

where $\models$ means that $E_{t,l}$ "explains" the predicted anomaly $P_{t,l}$. Specifically, Exathlon currently supports features $\mathcal{A}$ specified as interpretable names for the M feature columns of $P_{t,l}$ (i.e., ignoring its time dimension of l rows). For example, an explanation returned as a *logical formula* (e.g., by EXstream, MacroBase or Anchors) could take the form of the following conjunction of predicates:

$$E^{\text{formula}}_{t,l}(a_1, \ldots, a_M) = a_1 \geq 10 \wedge (a_2 \geq 30 \wedge a_2 \leq 50),$$

while an explanation returned as *feature importance scores* (e.g., by LIME or SHAP) could take the form of the following set of (feature, score) tuples:

$$E^{\text{scores}}_{t,l}(a_1, \ldots, a_M) = \{(a_3, 0.58), (a_1, 0.25), (a_2, 0.08)\}.$$

In the latter example, the explanation indicates that a_3, a_1 and a_2 were found as the features that most explain $P_{t,l}$, optionally after an internal thresholding step performed by the ED method (e.g., top-k selection or other filtering mechanisms, like in EXstream [90]).

Finally, we define an *extraction function* G over $E_{t,l}$, that returns the set of features $\mathcal{A}_{t,l}$ used in the explanation:

$$G(E_{t,l}(a_1, \ldots, a_M)) =: \mathcal{A}_{t,l} \subseteq \mathcal{A},$$

[3]The notations $P_{t,l}$ and $E_{t,l}$ do not relate to indexing. Rather, we consider P and E as objects characterized by their start index t and length l in $\boldsymbol{S}$.

along with the *size* of $E_{t,l}$, as the size of its feature set $\mathcal{A}_{t,l}$:

$$|E_{t,l}(a_1, \ldots, a_M)| = |\mathcal{A}_{t,l}|.$$

In our previous examples, explanations would have sizes 2 and 3, as the numbers of features appearing in the logical formula and having non-zero importance scores, respectively.

Evaluation Criteria

The two criteria we identified for evaluating ED functionality are based on whether explanations are considered for a single anomalous instance at a time (i.e., at a *local* level) or for a group of anomalous instances (i.e., at a *global* level):

- **ED1 (Local Explanation).** The local explanation of a (single) anomaly instance should provide a *compact* yet *meaningful* piece of information. In particular, the explanation produced should be *locally faithful* [78], meaning here that the same explanation should hold for immediate "neighbors" of the explained instance.
- **ED2 (Global Explanation).** Deriving an identical and succinct explanation for a large set of anomalous instances is usually not feasible. A "global" explanation is therefore usually composed of a set of instance-wise explanations (e.g., Ribeiro et al. [78] chose the most representative k instances to derive a "model" explanation from LIME). In this benchmark, we choose to group explanations based on the type of event they relate to (if available), to check whether these explanations are *consistent* or have any *predictive power* within the group.

Desired Properties and Evaluation Metrics

Exathlon evaluates both local and global explanations for three desired **properties**, reflected by quantitative **metrics**.

Conciseness Following the Occam's razor principle, humans favor smaller, and thus simpler explanations. To reflect this property, we consider the **Size** metric, defined as the number of features used in an explanation $E_{t,l}$ for ED1, and as the average number of features used across a set of N explanations $\{E_{t_i,l_i}\}_{i=1}^{N}$ for ED2[4]:

$$\boxed{\text{Size}_{\text{ED1}}(E_{t,l}) := |E_{t,l}| \quad , \quad \text{Size}_{\text{ED2}}(\{E_{t_i,l_i}\}_{i=1}^{N}) := \frac{1}{N}\sum_{i=1}^{N} |E_{t_i,l_i}|.}$$

Consistency Anomalies of the same type occurring in a similar context should have consistent explanations. For both ED1 and ED2, we only check for consistency using the set of features reported, without considering their numerical values.

For ED1, consistency translates to the notion of *stability*, which requires that an instance explanation stay similar when the input data is slightly altered. Multiple mechanisms can be considered to alter the input data[5]. In this work, alteration is performed through a subsampling procedure, generating a set of K samples $\{P_{t,l}^{(i)}\}_{i=1}^{K}$ from a predicted anomaly $P_{t,l}$, and deriving explanations $\{E_{t,l}^{(i)}\}_{i=1}^{K}$ for them. We then generalize the application of

[4]The explanations of such a set can technically belong to different sequences. We however assume in this section that all predicted ranges considered in ED2 belong to the same test sequence $\boldsymbol{S}$ to simplify the notation.

[5]These include "perturbations" of the input features (like for instance done in [78, 79]), which should be considered as an alternative option in the future, as they could be more suitable in some contexts, while also allowing the evaluation of stability when $l = 1$.

our extraction function to a set of explanations as the *duplicate-preserving* union of its outputs for each explanation in the set:

$$G(\{E_{t,l}^{(i)}(a_1,\ldots,a_M)\}_{i=1}^K) = \bigcup_{i=1}^K G(E_{t,l}^{(i)}) = \bigcup_{i=1}^K \mathcal{A}_{t,l}^{(i)} =: \mathcal{A}_{t,l}^{\cup}.$$

To reflect the stability property, we define our **instability** metric as the Shannon entropy of this bag of explanatory features:

$$\boxed{\text{Instability}(\{E_{t,l}^{(i)}\}_{i=1}^K) := H(\mathcal{A}_{t,l}^{\cup}) = -\sum_{i=1}^M p(a_i)\log_2 p(a_i),}$$

with the convention for $p(a_i) = 0$ that $0 \times \log_2(1/0) \equiv 0$ [107], and:

$$p(a_i) = \frac{1}{|\mathcal{A}_{t,l}^{\cup}|} \sum_{a \in \mathcal{A}_{t,l}^{\cup}} \mathbf{1}(a = a_i)$$

with $\mathbf{1}(a = a_i)$ the indicator function, outputting 1 if the feature a of the bag $\mathcal{A}_{t,l}^{\cup}$ is equal to the feature a_i, else 0. The smaller the entropy value, the less uncertain will be the outcome of randomly drawing an explanatory feature from the bag $\mathcal{A}_{t,l}^{\cup}$, and hence the more the features used to explain the different anomaly samples will agree with each other. In the ideal case, where all explanations $\{E_{t,l}^{(i)}\}_{i=1}^K$ are identical and of size Q, the entropy takes its minimum value of $H_Q = \log_2(Q)$. To alleviate the issue of the instability baseline being correlated with the size of explanations, we also report a *normalized* version of instability, defined as:

$$\boxed{\text{Norm-Instability}(E_{t,l}, \{E_{t,l}^{(i)}\}_{i=1}^K) := 2^{\text{Instability}(\{E_{t,l}^{(i)}\}_{i=1}^K)} / \text{Size}_{\text{ED1}}(E_{t,l}).}$$

Normalized instability (equal to $2^{\text{Instability}(\{E_{t,l}^{(i)}\}_{i=1}^K)} / 2^{\log_2(\text{Size}_{\text{ED1}}(E_{t,l}))}$) represents the ratio between the size of an explanation with same entropy as the combined explanations of the instance samples, and the actual size of the original instance's explanation.

For ED2, the consistency property translates to the concept of *concordance*, which requires that explanations for anomalies of the same type be consistent. To reflect it, we define the **discordance** metric similarly to instability, replacing the set of K subsampled instances $\{P_{t,l}^{(i)}\}_{i=1}^K$ by a set of N anomalies of the same type $\{P_{t_i,l_i}\}_{i=1}^N$ (and corresponding explanations $\{E_{t_i,l_i}\}_{i=1}^N$):

$$\boxed{\text{Discordance}(\{E_{t_i,l_i}\}_{i=1}^N) := \text{Instability}(\{E_{t_i,l_i}\}_{i=1}^N),}$$

with its corresponding normalized version:

$$\boxed{\text{Norm-Discordance}(\{E_{t_i,l_i}\}_{i=1}^N) := 2^{\text{Discordance}(\{E_{t_i,l_i}\}_{i=1}^N)} / \text{Size}_{\text{ED2}}(\{E_{t_i,l_i}\}_{i=1}^N).}$$

Accuracy The last (and hardest) property we consider in our benchmark is whether explanations can be *predictive* of "similar" anomaly instances (as defined above for ED1 and ED2). We evaluate this property by turning each explanation into a predictive model, and deriving accuracy metrics for this model's predictions around similar instances. As such, accuracy can only be assessed for ED methods that provide their explanations in a form that can be used for prediction (i.e., mapping a given data point to 0 or 1, $E_{t,l} : \mathbb{R}^M \to \{0,1\}$). ED methods that support accuracy evaluation for instance include those

that output logical formulas [90, 91, 79] or decision trees [84], but not those that output feature importance or SHAP scores [81]. Even among methods whose explanations can be used for prediction, the literature largely lacks ED methods that can provide explanations characterizing temporal patterns. For this reason, Exathlon defines the accuracy of an explanation as the *point-based* **Precision**, **Recall** and **F-score** it can achieve when used as a prediction model around other anomaly instances.

For ED1, accuracy metrics are computed for a given predicted anomaly $P_{t,l}$ by randomly splitting its data points into $P_{t,l}^{\text{train}}$ and $P_{t,l}^{\text{test}}$, generating an explanation $E_{t,l}^{\text{train}}$ for the anomaly points of $P_{t,l}^{\text{train}}$, and reporting its Precision and Recall as a predictive model on $P_{t,l}^{\text{test}}$ plus some neighboring normal data immediately preceding or following the anomaly instance. In practice, the reported Precision and Recall are obtained by averaging across K repetitions of this procedure, each time using a different random split[6].

For ED2, the Precision and Recall of an anomaly instance's explanation is computed on the anomalous and neighboring normal data of the other instances of the same type. This enables to assess whether the "rule" defined by the anomaly explanation can differentiate between normal and anomalous data around other similar instances.

3.2.3 Computational Performance

Along with functionality, Exathlon can also report the *computational performance* of both AD and ED methods for a given data dimensionality and cardinality.

Our computational performance properties **P1** and **P2** relate to anomaly detection. Their corresponding metrics are the **total training time** of the AD method and its **average inference time** on the test set, respectively. **P3** relates to explanation discovery, and is reflected in the benchmark as the **average explanation time** of the ED method.

3.3 Explainable Anomaly Detection Pipeline

This section presents the full explainable AD pipeline for multivariate time series we provide to facilitate the usage and configuration of the benchmark, as well as the implementation of new AD and ED methods. This end-to-end pipeline consists in a sequence of **configurable steps** ranging from data partitioning to data transformation to anomaly detection and explanation discovery. We implemented it using Python, with some components relying on popular libraries like Scikit-Learn [108], TensorFlow [109] and PyTorch [110]. This pipeline was designed to be open and modular, within and beyond our particular use case, allowing different datasets and methods to be added and combined. Figure 3.3 provides an overview.

3.3.1 Data Partitioning

The data partitioning step takes as input the path to the raw sequences data and labels. A dataset-specific connector first performs data loading, parsing and selection, as well as any necessary data cleaning. This step then partitions data into a set of N_1 and N_2 intermediate training and test sequences, $\mathcal{S}_{\text{train}}^{\text{inter}}$ and $\mathcal{S}_{\text{test}}^{\text{inter}}$, with corresponding (final) sequences of labels $\mathcal{Y}_{\text{train}}$ and $\mathcal{Y}_{\text{test}}$:

[6] It should be noted that Exathlon's implementations of ED1 accuracy and consistency currently assume that an anomaly's ground-truth explanation *remains constant over time*. In practice, it can however happen that long-lasting anomalies display distinct "phases" of different nature (e.g., subtle, precursory signals of a phenomenon followed by the phenomenon in itself), in which case we cannot expect random anomaly subsamples to yield consistent, succinct explanations. For such anomalies, one could employ domain knowledge to treat distinct phases separately (e.g., only triggering ED on the first minute of AD predictions, considering this phase as the most interesting to explain for deploying corrective actions).

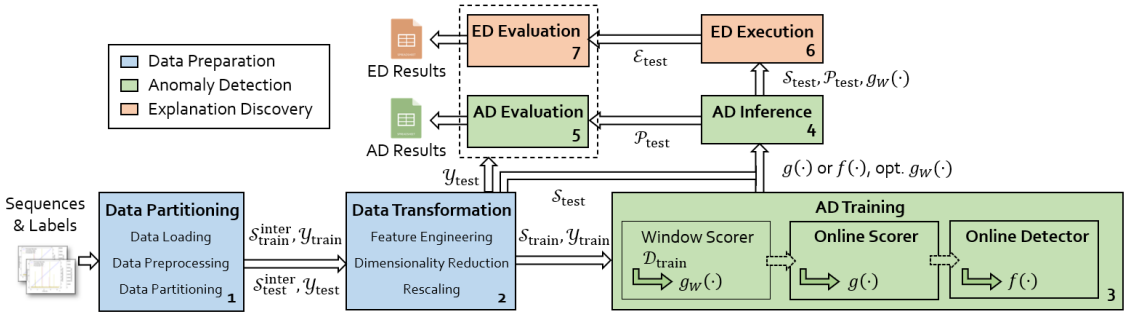


Figure 3.3: A pipeline for explainable anomaly detection over multivariate time series.

$$\mathcal{Y}_{\text{train}} = \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(N_1)}\} \quad , \quad \mathcal{Y}_{\text{test}} = \{\boldsymbol{y}^{(N_1+1)}, \ldots, \boldsymbol{y}^{(N_1+N_2)}\},$$

with $\boldsymbol{y}^{(i)} \in [0 \,..\, C]^T$, such that $y_t^{(i)} = c$ if the record at index t in sequence i is of class $c \in [0 \,..\, C]$ (in Exathlon, we define $c = 0$ for *normal* and $c > 0$ for *anomalous*). Although not shown in Figure 3.3 nor in our description, this step also allows to consider part of the training set as *validation* sequences, whose data can for instance be used for early stopping and checkpointing by deep learning methods. Both training and validation labels are optional and never considered by unsupervised methods.

3.3.2 Data Transformation

The data transformation step takes in the intermediate training and test sequences from the previous step and turns them into the N_1 training sequences and N_2 test sequences considered by the AD and ED methods:

$$\mathcal{S}_{\text{train}} = (\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(N_1)}) \ , \ \mathcal{S}_{\text{test}} = (\boldsymbol{S}^{(N_1+1)}, \ldots, \boldsymbol{S}^{(N_1+N_2)}),$$

where each $\boldsymbol{S}^{(i)}$ consists of T ordered data records[7] of dimension M. Depending on the user's needs, this transformation process can involve configurable components arranged in any order, for instance including data-specific *feature engineering*, general *dimensionality reduction* (e.g., Principal Component Analysis [56], Factor Analysis [111]) and/or *data rescaling* (e.g., standardization, normalization).

3.3.3 Anomaly Detection Training

The goal of the anomaly detection training step is to produce an AD method in the form of either one of two sequence-to-sequence functions g or f. We refer to g as a *record scoring function*. It maps a sequence $\boldsymbol{S}$ to a sequence of record-wise real-valued *anomaly scores*:

$$\begin{aligned} g : \mathbb{R}^{T \times M} &\rightarrow \mathbb{R}^T \\ \boldsymbol{S} &\mapsto g(\boldsymbol{S}), \end{aligned}$$

where a higher score corresponds to more "abnormality" for the method. In contrast, the function f, which we call a *record detection function*, maps a sequence $\boldsymbol{S}$ to a sequence of record-wise *binary predictions*:

[7] In practice, Exathlon supports sequences of different lengths T_i, but we consider a single T here to simplify the notation.

$$f : \mathbb{R}^{T \times M} \to \{0,1\}^T$$
$$\boldsymbol{S} \mapsto f(\boldsymbol{S}).$$

The expected output of this training step is therefore flexible to the type of method employed. In this work, we always consider the setting of **offline training** and **online inference**, where training has to be performed offline on $\mathcal{S}_{\text{train}}$, and inference on $\mathcal{S}_{\text{test}}$ only considering the *previous* data for a given record at time index t:

$$g(\boldsymbol{S}^{(i)})_t = g(\boldsymbol{S}^{(i)}_{1:t})_t$$
$$f(\boldsymbol{S}^{(i)})_t = f(\boldsymbol{S}^{(i)}_{1:t})_t \ , \ \forall t \in [1 \,..\, T] \ , \ i \in [N_1 \,..\, N_1 + N_2].$$

For this reason, and although online inference is not *required* by the pipeline, we always refer to the AD methods based on g and f as **online scorers** and **online detectors**, respectively.

Window Scorers, Online Scorers and Online Detectors

Both online scorers and detectors can optionally rely on a **window scorer**. In this case, the online scorer (or detector) definition and/or training is preceded by additional **data windowing** and **window scorer training** steps in the pipeline.

For data windowing, we employ a *windowing operator* W_L, extracting and labeling sliding windows, or *samples*, of length $L > 0$ from a given sequence i:

$$W_L(\boldsymbol{S}^{(i)}, \boldsymbol{y}^{(i)}) = \{(\boldsymbol{S}^{(i)}_{t-L+1:t}, \text{mode}(\boldsymbol{y}^{(i)}_{t-L+1:t})\}_{t=L}^{T} =: \{(\boldsymbol{X}^{(i)}_t, \tilde{y}^{(i)}_t)\}_{t=L}^{T},$$

with $\boldsymbol{X}^{(i)}_t \in \mathbb{R}^{M \times L}$ the samples, and $\tilde{y}^{(i)}_t \in [0 \,..\, C]$ the corresponding labels, indicating their *majority class*:

$$\text{mode}(\boldsymbol{y}_{t-L+1:t}) = \underset{c \in \boldsymbol{y}_{t-L+1:t}}{\arg\max} \sum_{t'=t-L+1}^{t} \mathbf{1}(y_{t'} = c).$$

Window scorers are then trained on samples extracted from the training sequences and labels[8]:

$$\mathcal{D}_{\text{train}} := \bigcup_{i \in [1..N_1]} \left\{ W_L(\boldsymbol{S}^{(i)}, \boldsymbol{y}^{(i)}) \right\}.$$

to derive a *window scoring function*, that assigns an anomaly score to a given window, proportional to its abnormality for the method:

$$g_W : \mathbb{R}^{L \times M} \to \mathbb{R}$$
$$\boldsymbol{X} \mapsto g_W(\boldsymbol{X}).$$

Online scorers (and detectors) can be derived from g_W independently from the anomaly detection method used, which is the strategy we adopt in the rest of this work. Specifically, our experiments compare all AD methods in the same unifying framework, turning *window scorers* into *online scorers* by setting for a given test sequence $\boldsymbol{S}$ and $\beta \in [0, 1)$:

[8] Although extracted based on sequence labels, window labels are never used by unsupervised methods.

$$g(\boldsymbol{S}; L, \beta)_t = \begin{cases} -\infty & \text{if } t < L \\ (1-\beta)\hat{y}_L =: m_L & \text{if } t = L \\ \frac{\beta m_{t-1} + (1-\beta)\hat{y}_t}{(1-\beta^{t+1})} =: m_t & \text{if } t > L \end{cases}$$

With:

$$\hat{y}_t = g_W(\boldsymbol{S}_{t-L+1:t}) \ , \ \forall t \in [L \,..\, T].$$

For a test sequence, we therefore start by assigning an anomaly score to the current sliding window of length L using g_W (which is fixed and trained offline). We then make this window score correspond to the anomaly score of its last record (i.e., the record we just received in an online setting). To allow some additional control regarding the tradeoff between the "stability" and "reactivity" of the record scoring function[9], we further apply an *exponentially weighted moving average* (EWMA) with smoothing hyperparameter β to the anomaly scores. This produces the final output of the record scoring function g for a test sequence, prepended with infinitely low anomaly scores for the timestamps before a first full window of length L is received. When using this framework, both L and β are hyperparameters to set for every anomaly detection methods.

The Exathlon pipeline also allows to turn *online scorers* into *online detectors* through **threshold selection**. If training labels are available, threshold selection can be performed in a *supervised* way by maximizing a given performance metric (e.g., F1-score). Otherwise, it can be performed in an *unsupervised* way, typically modeling the distribution of (finite) anomaly scores assigned by g in training sequences, and setting the threshold value to the start of this distribution's upper tail.

3.3.4 AD Inference and Evaluation

The AD inference and evaluation steps evaluate online scorers or detectors with respect to the requirements presented in Section 3.2.1. Specifically, online scorers g are evaluated based on their ability to separate normal from anomalous records in anomaly score space, leaving the selection of a suitable threshold to human operators when the solution is deployed in practice [103]. A way to do so is to consider every possible detector f that can be derived from g using a fixed anomaly score threshold. That is, given the set of all anomaly scores g assigned in test sequences:

$$\hat{\mathcal{G}}_{\text{test}} := \bigcup_{i \in [N_1+1 \,..\, N_1+N_2]} \left\{ \left\{ g(\boldsymbol{S}^{(i)}; L, \beta)_t \ , \ t \in [1 \,..\, T] \right\} \right\},$$

consider all detectors $\mathcal{F} = \{f(\cdot; L, \beta, \delta) \ , \ \delta \in \hat{\mathcal{G}}_{\text{test}}\}$, where the binary record-wise prediction assigned by a detector $f(\cdot; L, \beta, \delta)$ in a sequence $\boldsymbol{S}$ at time t is defined as:

$$f(\boldsymbol{S}; L, \beta, \delta)_t := g(\boldsymbol{S}; L, \beta)_t > \delta.$$

Plotting the Precision and Recall for every such detector on the test set gives the Precision-Recall (PR) curve. The pipeline supports the Area Under the Precision Recall Curve (**AUPRC**) as an online scorer metric, but also the Precision, Recall and F1-score achieved by the detector $f(\cdot; L, \beta, \delta^*)$, where δ^* is the anomaly score threshold that gave the maximum F1-score on the test set (i.e., the "best" point on the PR-curve). We refer to this latter metric as the **peak F1-score** achieved by the online scorer g, indicating the detection performance this scorer would achieve given the adequate threshold.

[9] This tradeoff is also influenced by the window length L, the window scoring function and the types of anomalies.

The pipeline also supports running and evaluating online detectors f using the range-based Precision, Recall and F-score metrics directly. In particular, performing inference with an online detector f produces the predicted ranges introduced in Section 3.2.2 and required by the ED inference and evaluation steps:

$$\mathcal{P}_{\text{test}} := \bigcup_{i \in [N_1+1\,..\,N_1+N_2]} \left\{ \left\{ P^{(i)}_{t_j,l_j} \ , \ j \in [1\,..\,K_i] \right\} \right\},$$

with K_i the number of anomalous ranges predicted by f in sequence $\boldsymbol{S}^{(i)}$.

3.3.5 ED Execution and ED Evaluation

The ED execution step produces explanations $\mathcal{E}_{\text{test}}$ for the ranges $\mathcal{P}_{\text{test}}$ predicted by an online detector f in test sequences. Like mentioned in Section 2.3, the Exathlon pipeline supports two main types of ED methods, referred to as **model explainers** and **data explainers**. The goal of model explainers is to explain the prediction process of a particular *model*, which makes them rely, at least, on this model's decision function. Specifically, the pipeline currently supports the evaluation of *window scorer* explainers, explaining a mapping g_W from an input window to its real-valued anomaly score. In contrast, data explainers seek to explain the differences between an anomalous and a reference *dataset*, only relying on anomaly detection methods to define these datasets through their prediction labels. By default, the pipeline defines the reference dataset of an anomaly range as the normal range that immediately precedes it in time and spans twice its length.

The ED evaluation step takes in the $\mathcal{E}_{\text{test}}$ explanations produced by ED execution and evaluates them based on the criteria defined in Section 3.2.2. By default, this step computes ED1 instability by subsampling 80% of the explained instances five times, and ED1 accuracy across five folds using 80/20 splits. Since ED2 requires the event type information, the pipeline currently supports setting $\mathcal{P}_{\text{test}}$ as the ground-truth anomaly ranges directly (i.e., considering human annotations), as well as the intersection between the AD method's predictions and ground-truth anomalies (i.e., considering true positives only, for which we know the expected type)[10]. The former scenario also allows diagnosing data explainers without the need for an AD method at all.

3.4 LEADS Viewer

This section presents the Viewer of Leads via an Explainable Anomaly Detection System (LEADS Viewer)[11], a platform we designed and released to visualize high-dimensional time series along with AD and ED predictions[12].

LEADS Viewer can be used as a tool for qualitatively assessing the performance of explainable AD methods, either jointly with or independently of the quantitative benchmarking provided by Exathlon. Launching the platform brings us to a home page, from which we can create or load an existing project. After selecting a project, we get to upload *sequences*, each corresponding to a multivariate time series, along with optional anomaly detection and explanation results. The uploaded sequences appear with their name in a `SEQUENCE` tab. They should all have the same feature names, from which we get asked to specify a list of Key Performance Indicators (KPIs). The values of these

[10] In its current implementation, our ED2 evaluation criteria therefore assumes the ED method operates on top of a working AD method, with most of its predictions being true positives, and very few of them being (ignored) false positives.

[11] The name comes from the aim of providing "leads", or "clues" toward characterizing abnormal behaviors of entities from collected data.

[12] The implementation of LEADS Viewer was carried out by Jia Li, Ran Wang and Junqiang Chen, with whom continuous exchanges also helped refine the design.

KPIs will be displayed as time plots on the left of the DASHBOARD tab for every sequence, and can optionally be ignored in explanations. In practice, KPIs should reflect the overall "health" or "status" of a recorded entity, typically corresponding to the few metrics that human operators would watch to check its proper behavior. Selecting a sequence from the SEQUENCE tab sends us to its corresponding DASHBOARD tab, displaying the feature values and optional predictions for this sequence.

Figure 3.4 shows an example of LEADS Viewer's DASHBOARD tab for the T1 (Bursty Input) trace 5_1_500000_62 of Exathlon's Spark Streaming dataset. In this example, we can see we selected the application's last completed batch total delay, scheduling delay and processing delay features as KPIs, appearing as time plots on the left side of the dashboard. The total delay for a batch of streaming records corresponds to the sum of its scheduling delay and processing delay, with the scheduling delay being the time elapsed between the arrival of this batch and the start of its processing, and the processing delay its total processing time. The other 234 features from Section 4.2.4's feature set are displayed on the right side of the dashboard, also as time plots by default, but with the option of viewing them in a tabular format instead. On top of the features, Figure 3.4a shows the anomaly ranges predicted by our detection method in red (CEADAL from Chapter 6 in this case, only considering the first 120 seconds of its predicted ranges, and removing the ranges smaller than 60 seconds as post-processing). From Figure 3.4b, we can see that double-clicking on a specific predicted range provides a zoomed-in view around it, and displays its corresponding explanation if available. In this example, we explain the first predicted range of trace 5_1_500000_62 using an internal version of EXstream [90], providing both feature importance scores and decision rules in the form of conjunctions of predicates (the most comprehensive explanation format currently supported by the platform). As we can see, the explanation returned states that the numbers of records in the last received batch (i.e., received in the last five seconds, given the batch interval set for the applications) are abnormally high, fluctuating around 12M instead of the usual 3M. This corresponds to an abnormally high rate of data records coming from the sender, and therefore constitutes a plausible explanation for a T1 (Bursty Input) event. Additional information about the supported data formats, functionalities, and usage of LEADS Viewer is available at https://github.com/exathlonbenchmark/leads-viewer.

LEADS Viewer could be compared to the MTV platform [112] recently released within the Signal Intelligence (Sintel) project [32]. Like MTV, LEADS Viewer proposes to visualize the predictions of an anomaly detection method over a "multi-signal view" of multivariate time series data. While LEADS Viewer does not currently include some of MTV's features, like its "multi-aggregation" view (i.e., simultaneous display of signals at different aggregation levels), anomaly annotations or discussion panels, it introduces additional concepts like KPIs and the visualization of anomaly explanations, along with an optional tabular view that can be useful for various use cases, such as fraud detection in financial transactions. Overall, LEADS Viewer and MTV were originally designed to serve different purposes, and could thus be seen as complementary. Specifically, MTV centers around time series anomaly detection for teams, mostly in industrial settings, while LEADS Viewer was built as a lightweight tool for researchers and practitioners to quickly visualize their time series data and explainable anomaly detection outputs.

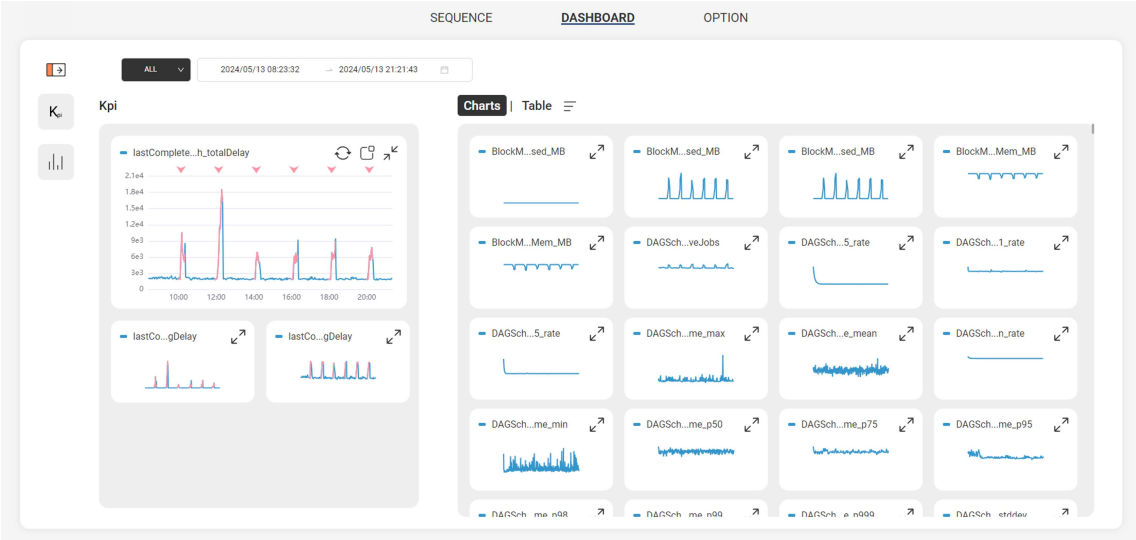
3.5 Summary and Conclusions

In this chapter, we presented Exathlon, the first public benchmark for explainable anomaly detection over high-dimensional time series. We introduced its (i) Spark Streaming dataset and use case, (ii) evaluation methodology for anomaly detection (AD) and explanation discovery (ED), and (iii) modular pipeline for explainable anomaly detection. We also intro-

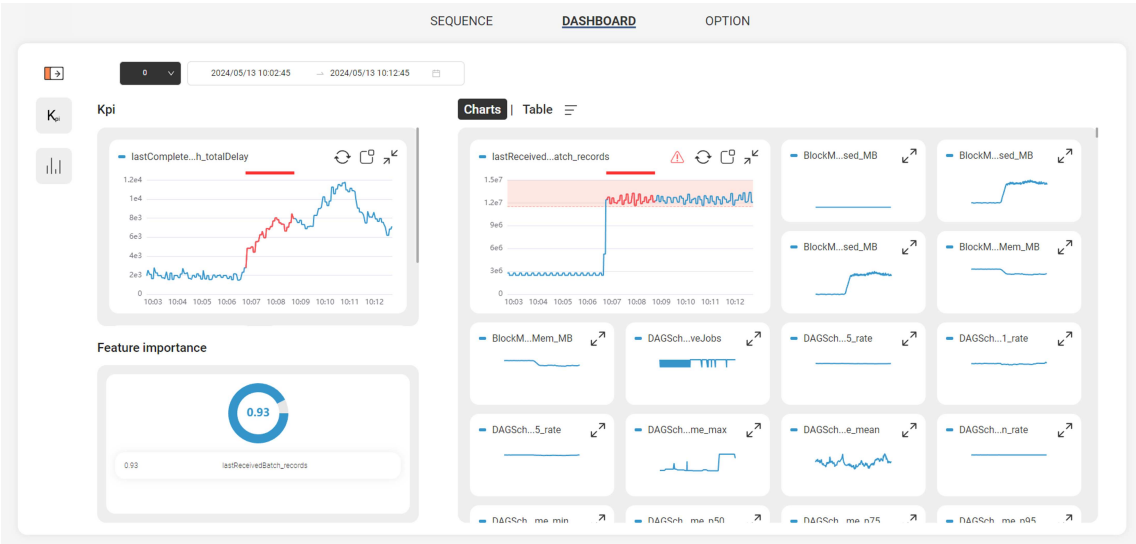
duced LEADS Viewer, a graphical user interface we released to visualize high-dimensional time series and further diagnose predictions of AD and ED methods.

We released Exathlon and LEADS Viewer to address the lack of development and benchmarking tools available to academic research for explainable anomaly detection in high-dimensional time series. This problem is a central component of Exathlon's Spark Streaming (AIOps) use case, which also comes with other specific challenges that we hope these open benchmarking tools will help study more and address.

In the next chapter, we will conduct an experimental study of representative unsupervised AD methods on Exathlon's dataset. Throughout this study, we will come back to these specific AIOps challenges, showing the way they are reflected in our experimental setup, and the limitations of traditional methods to address them.



(a) Features time plots and CEADAL anomaly predictions.



(b) EXstream's explanation of the first predicted anomaly.

Figure 3.4: LEADS Viewer's `DASHBOARD` tab for the T1 (Bursty Input) trace `5_1_500000_62` of Exathlon's Spark Streaming dataset, with CEADAL and EXstream predictions.

4 Unsupervised Anomaly Detection Study

In this chapter, we conduct an in-depth **benchmarking analysis** of representative unsupervised anomaly detection (AD) methods against the Exathlon benchmark and dataset introduced in the previous chapter. We start by formalizing the problem statement considered in Section 4.1. We then describe our experimental setup in Section 4.2, corresponding to the parameters used for the steps and substeps of the Exathlon pipeline. We follow by detailing the characteristics of the Exathlon dataset in Section 4.3, including the specific way AIOps challenges are reflected in our experimental setup, as well as the connection between the dataset's event types and the anomaly types commonly defined for time series. Section 4.4 presents the unsupervised methods we included in this study, whose performance is reported and analyzed in Section 4.4. In this last section, we further highlight the 15 conclusions drawn from this study as **C1-15**, including the tradeoffs and main **limitations** we found for the considered methods to address our AIOps **challenges**.

4.1 Problem Statement

This section formalizes the anomaly detection problem we want to solve in this unsupervised setting.

We consider N_1 training sequences and N_2 test sequences:

$$\mathcal{S}_{\text{train}} = (\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(N_1)}) \ , \ \mathcal{S}_{\text{test}} = (\boldsymbol{S}^{(N_1+1)}, \ldots, \boldsymbol{S}^{(N_1+N_2)}),$$

where each $\boldsymbol{S}^{(i)}$ consists of T ordered data records of dimension M. For each *test* sequence, we further consider a sequence of *anomaly labels*:

$$\mathcal{Y}_{\text{test}} = \{\boldsymbol{y}^{(N_1+1)}, \ldots, \boldsymbol{y}^{(N_1+N_2)}\},$$

with $\boldsymbol{y}^{(i)} \in \{0, 1\}^T$, such that:

$$\begin{cases} y_t^{(i)} = 1 & \text{if the record at index } t \text{ in sequence } i \text{ is } \textit{anomalous}, \\ y_t^{(i)} = 0 & \text{otherwise (i.e., the record is } \textit{normal}). \end{cases}$$

Our goal is to build an anomaly detection model in the form of a *record scoring function* $g : \mathbb{R}^{T \times M} \rightarrow \mathbb{R}^T$, mapping a sequence $\boldsymbol{S}$ to a sequence of real-valued record-wise anomaly scores $g(\boldsymbol{S})$, which assigns higher anomaly scores to anomalous records than to normal records in test sequences. That is:

$$g(\boldsymbol{S}^{(i)})_{t_1} > g(\boldsymbol{S}^{(j)})_{t_2} \ , \ \forall i, j \in [N_1 \,..\, N_1 + N_2] \ , \ t_1, t_2 \in [1 \,..\, T] \text{ s.t. } y_{t_1}^{(i)} = 1 \wedge y_{t_2}^{(j)} = 0.$$

This record scoring function should further be constructed in a setting of *offline training* and *online inference*, meaning training has to be done offline on $\mathcal{S}_{\text{train}}$, and inference has to be done on $\mathcal{S}_{\text{test}}$ only considering previous data for a given record at time index t:

$$g(\boldsymbol{S}^{(i)})_t = g(\boldsymbol{S}_{1:t}^{(i)})_t \ , \ \forall i \in [N_1 \,..\, N_1 + N_2] \ , \ t \in [1 \,..\, T].$$

We note that this problem statement encompasses both a point-based requirement and the AD1-4 criteria presented in Section 3.2.1, in the sense that solving it would perfectly satisfy all of them for a record scoring function g.

4.2 Experimental Setup

This section describes the experimental setup we consider to run and evaluate AD methods on Exathlon's dataset, with respect to the problem statement of Section 4.1. This description corresponds to the parameters we set for the corresponding steps (and substeps) of the Exathlon pipeline presented in Section 3.3.

4.2.1 Data Selection

In all our experiments, we never consider the Spark Streaming applications 7 and 8, for which there are no disturbed and undisturbed traces, respectively. In practice, one could also consider pruning the first few and/or last minutes of data for every trace, to account for various recording artifacts, along with any specific noisy data segments we identified in both training and test traces. We however choose not to remove any other data item, in order to study the behavior of methods with respect to such minority and noisy patterns.

Our primary goal with this use case is to detect anomalies in the behavior of a running Spark Streaming *application*, as opposed to the behavior of the entire four-node *cluster* an application is running on. As such, we always remove from our labels the CPU contention events that had no impact on a recorded application's components (i.e., that occurred on nodes where this application had no running driver or executors). In practice, we label those anomalies the same way as other "*unknown*" anomalous events, in order not to penalize CPU contention Recall for missing them, nor Precision for detecting records as abnormal in these ranges.

4.2.2 Data Preprocessing

Like mentioned in Section 3.1, the metrics collected for Spark Streaming traces allocated 140 columns for each of five "executor spots" in the data, saved in case one of the two to three active executors of an application failed during its execution. In practice, inactive executor spots in the data took the default value of -1 (as a placeholder for "null"). This value of -1 was however also (and mainly) used to refer to (a potential subset of) "missing" metrics, not received fast enough for the expected timestamp during data collection. This convention yielded two types of contiguous "-1 ranges" for executor metrics in the data, with some meaning the executor was inactive, and some meaning it was not reachable by data collection (typically, but not only, during anomalies). For every executor `e`, we distinguish these two cases based on the `{e}_executor_runTime_count` counter metric. Specifically, if a -1 range occurs between two non-(-1) ranges and this counter metric was reset after it, then this -1 range corresponds to an "inactive executor range". Otherwise, it corresponds to a "missing range". We handle cases of starting and ending -1 ranges through a combination of manual inspection and duration rules.

With these types of ranges distinguished, missing values were filled by propagating forward preceding valid values (or propagating backward following valid values when no such values existed). Inactive executor ranges were left as -1.

After this preprocessing, all metrics in the data should be either positive or -1. This was sometimes not the case for three specific metrics, which could mistakenly take the opposite of their "true value" due to other related metrics being -1. For this reason, we also set all negative metrics different from -1 to their opposite values.

Finally, to handle duplicate and missing timestamps, we resampled the data from all traces to match their supposed sampling period of one second (using the $\max(\cdot)$ aggregation function).

4.2.3 Data Partitioning

In this study, we consider the setup of building a single AD method instance for all the Spark Streaming applications in training, as opposed to building a distinct instance per application. We retained this setup to reduce the modeling cost as the number of applications increases, which relates to the AIOps challenge of being able to handle different contexts at scale. Besides, training a single model for a variety of entities is often considered more effective in practice, allowing this model to share knowledge across entities, and thus increasing data efficiency per entity [113]. We however do not require AD methods to generalize to *unseen* applications, by making sure the eight applications left after data selection are all represented in the training and test sequences.

Specifically, we define our training sequences as most of the undisturbed traces, plus some disturbed traces to increase the variety in application *settings* and *input rates* for the methods to learn from. We also make sure both the training and the resulting test disturbed traces cover all the event types of Exathlon (but remove anomalous data for the undisturbed methods). After data partitioning, our test sequences contain:

- **15** Bursty Input (**T1**) ranges.
- **5** Bursty Input Until Crash (**T2**) ranges.
- **6** Stalled Input (**T3**) ranges.
- **7** CPU Contention (**T4**) ranges.
- **5** Driver Failure (**T5**) ranges.
- **5** Executor Failure (**T6**) ranges.

4.2.4 Feature Engineering

All our compared methods consider the same features as input, built from an automated feature engineering step simply consisting in:

- Dropping the features collected by `nmon` (since these features reflect the behavior of the entire four-node cluster, sometimes unrelated to the application run represented in the trace).
- Dropping the features that were constant throughout the whole data.
- Differencing *cumulative* features (i.e., that were only increasing within a given trace).
- Averaging corresponding Spark executor features across "active executor spots" (non-(-1) after data preprocessing) into a single block of 140 features.

After this feature engineering, we get $M = 237$ features to use by the AD methods, which can be decomposed as follows:

- **168 Driver Features**
  - 18 "streaming" features. For example:
    - The processing delay and scheduling delay of the last completed batch.
    - The number of records in the last received batch.
  - 5 block manager features. For example:
    - The disk space used by the block manager.
    - The memory used by the block manager.
  - 32 JVM features. For example:
    - The heap memory usage of the driver.
    - The survivor space usage of the driver (the survivor space is a memory pool that holds objects having survived a young generation garbage collection, before those objects potentially get promoted to old generation memory).

  - 19 DAG scheduler features. For example:
    - The number of active jobs.
    - The number of running stages.
  - 94 live listener bus features. For example:
    - The number of messages received from the DAG scheduler in the last 1, 5 and 15 minutes.
    - The average processing time of messages received from the DAG scheduler.
- **69 Executor Features (Averaged Across Active Executors)**
  - 27 "executor" features. For example:
    - The CPU time.
    - The number of active tasks.
    - The number of bytes read and written to HDFS.
  - 38 JVM features, similar to those of the driver.
  - 4 netty block transfer features. For example:
    - The direct memory used by the shuffle client and server of the netty network application framework (sending and receiving blocks of data).
    - The heap memory used by the shuffle client and server of the netty network application framework.

4.2.5 Data Windowing

To compare different AD methods in a unified manner, this study relies on the framework introduced in Section 3.3.3, only considering methods that are both trained and used on data windows (sometimes called *samples* in the following), with anomaly scores derived from a window scoring function g_W. We therefore perform a data windowing step, producing sliding windows of length $L = 1$ for methods that model individual data records (called *point modeling* methods in the following), and $L = 20$ for sequence modeling methods.

Once sliding windows have been created from the training sequences, we further *balance* them by according to the (application, Spark settings, input rate) triplet they relate to. Since there is no reason for AD methods to favor any particular values of those aspects, we indeed ensure every combination that exists in the training data is equally represented. We make sure this process preserves the data cardinality, by randomly undersampling the over-represented combinations, and randomly oversampling the under-represented ones.

4.2.6 Evaluation Strategy

To simplify this study, we only consider *point-based* anomaly detection performance. Like in [103], we favor that AD methods induce a single, high-performing threshold over many "medium" ones, and thus evaluate AD methods using the **peak F1-score** metric of the Exathlon benchmark introduced in Section 3.3.4. When computing the F1-scores, we average Recall values across the different event types, considering them as equally important to detect no matter their cardinality in test data.

We benchmark anomaly detection methods assuming a purely unsupervised scenario, where labels are not assumed available even for tuning hyperparameters. As such, we report the performance of each method as its full box plot of peak F1-scores achieved across a "sensible" grid of hyperparameter values, like advised for instance in [61].

For every AD method, our window-based methodology to derive the record scoring function can induce "rightfully large" anomaly scores assigned to the $L-1$ records immediately following an anomaly (since the windows of length L used to compute them are partially anomalous). This may introduce some rightful "lags" in the anomaly predictions, hindering the global performance despite the method behaving properly. The recently proposed

Volume Under the Surface (VUS) metric [114] alleviates this issue, and could be considered to extend this study in the future. In our evaluation, we simply ignore the $L-1$ records following each test anomaly, where L is the window length used by the evaluated AD method.

When deriving our record scoring functions g, we always consider the following grid of anomaly score smoothing factors:

$$\beta \in (0, 0.8, 0.9, 0.95, 0.96667, 0.975, 0.98, 0.98333, 0.9875, 0.99167, 0.99375, 0.995),$$

which corresponds to considering approximately the last:

$$n_\beta \in (1, 5, 10, 20, 30, 40, 50, 60, 80, 120, 160, 200)$$

anomaly scores in the exponentially weighted moving average (with $n_\beta = 1/(1-\beta)$).

4.3 Data Characteristics

This section describes the characteristics of Exathlon's dataset under the experimental setup of Section 4.2. We first discuss the impact each *event type* could have on our extracted features, mapping them to the *anomaly types* commonly defined for time series in the literature [18, 115]. We then provide additional details about the diversity and shift in normal behaviors present in this dataset.

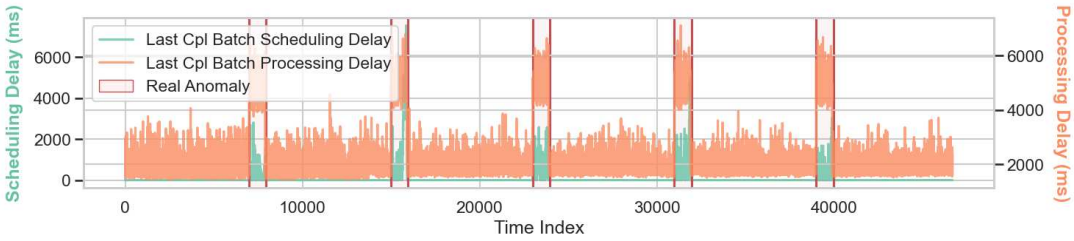
4.3.1 Event and Anomaly Types

Figure 4.1 shows an example disturbed trace for each type in Exathlon's dataset, projected on the application's last completed batch scheduling delay and processing delay features. The scheduling delay for a batch of streaming records corresponds to the time elapsed between the arrival of this batch and the start of its processing, while the processing delay refers to its complete processing time. The scheduling delay should ideally remain stable at either zero or a low value over time (i.e., batches keep getting processed shortly after they arrive). An increase in scheduling delay typically occurs when the processing delay exceeds the batch interval set for the application (i.e., batches arrive faster than they can be processed). This typically makes the scheduling and processing delays relevant Key Performance Indicators (KPIs) to consider when monitoring a Spark Streaming application.

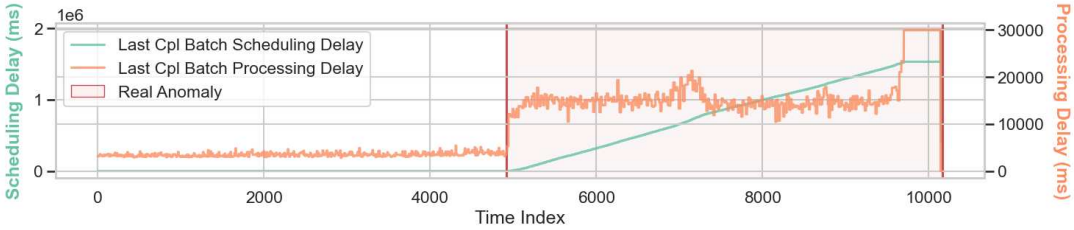
In the literature, anomalies in univariate time series have commonly been categorized into the following three types [18, 115]:

- **Point Anomalies.** Data records that significantly deviate from the rest of the data (e.g., a substantial extremum in value, which is generally the easiest type of anomaly to detect [18, 19]).
- **Contextual Anomalies.** Data records that are normal in a given context, but anomalous in another (e.g., a temperature of 5°C is normal in the winter in France, but anomalous in the summer).
- **Collective (or Sequence) Anomalies.** Data records that are normal when taken individually, but anomalous when considered as a sequence (e.g., recording the exact same temperature value over a long period of time can indicate a sensor malfunction period, even if the individual temperatures are normal).

As we will see, these three anomaly types are represented through the event types and features of the Exathlon dataset after our feature engineering.

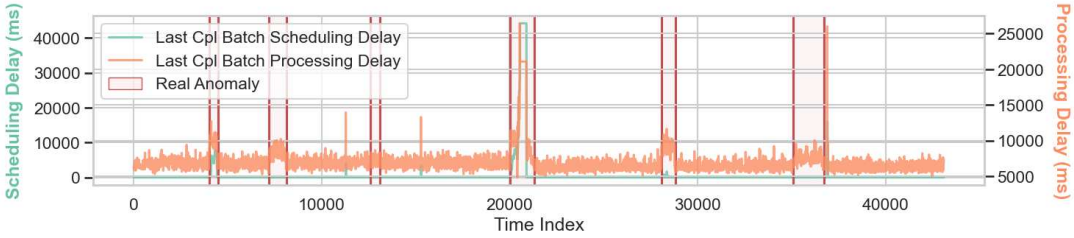


(a) Bursty Input (T1) trace 6_1_500000_65.

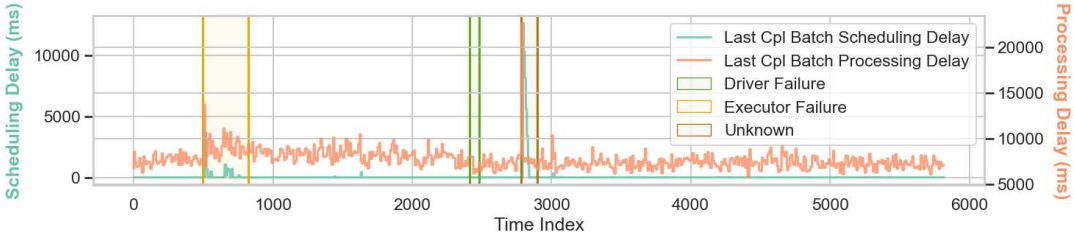


(b) Bursty Input Until Crash (T2) trace 10_2_1000000_67.

(c) Stalled Input (T3) trace 10_3_1000000_75.

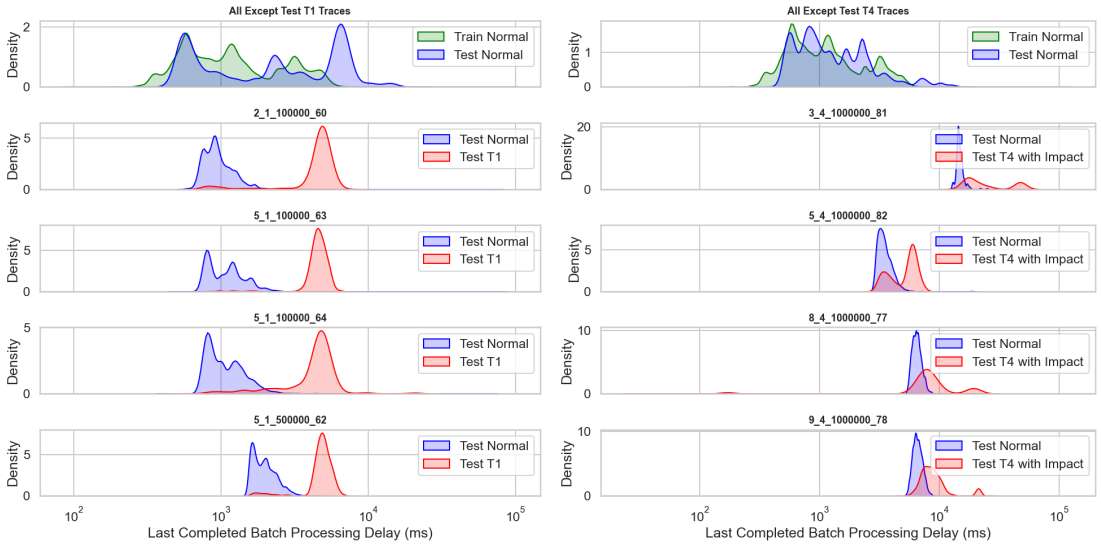


(d) CPU Contention (T4) trace 9_4_1000000_78.



(e) Process Failure (T5) trace 9_5_1000000_84.

Figure 4.1: Example disturbed trace of each type projected on the last completed batch scheduling and processing delays.



(a) Bursty Input (T1) traces.

(b) CPU Contention (T4) traces.

Figure 4.2: Kernel Density Estimate (KDE) plots of the last completed batch processing delay for training normal data, test normal data and test anomalous data.

Bursty Input (T1) and Bursty Input Until Crash (T2) Events

Figure 4.1a shows a time plot of a Bursty Input (T1) trace. Although T1 events could induce (imperfect) point anomalies for the scheduling delay feature, they were reflected by **contextual** anomalies for a lot of others, including the processing delay. To illustrate this, Figure 4.2a shows Kernel Density Estimate (KDE) plots of the processing delay values taken by training normal records, test normal records and test T1 records (i.e., belonging to T1 anomaly ranges). A separate subplot is shown for each test T1 trace, with an additional subplot for all of the remaining data in our experimental setup. All subplots share the same x-axis in log scale. From this figure, we can see that, although T1 events induce higher processing delays than normal *within the context of a trace*, these "higher" values actually appear normal with respect to the training and test normal data *globally* (i.e., taken from all traces together). In other terms, T1 events induce *contextual* anomalies for the processing delay feature, where the context is defined as the characteristics of a trace (further detailed in Section 4.3.2).

Figure 4.1b shows a time plot of a Bursty Input Until Crash (T2) trace. T2 events can be seen as "simpler" versions of T1 events. Since they are more sustained and eventually lead to an application crash, the anomalies induced by T2 events tend to become more "point-based" as time progresses within a trace.

Stalled Input (T3) Events

Figure 4.1c shows a time plot of a Stalled Input (T3) trace. T3 events can be seen as slightly more subtle than T1 and T2 events, in the sense that a lack of incoming data does not induce any delays or resources shortage for an application. Some aspects get easier to detect, like some point anomalies induced for the processing delay (which drops around its global minimum of zero). Other aspects however get more complex, like the **collective** anomalies induced for the "difference in total number of received records" feature. To illustrate this collective component, Figure 4.3 shows time plots of this feature for a normal and an anomalous subsequence of lengths 100 in T3 trace `10_3_1000000_75` (sharing the same y-axis). The "total number of received records" being a cumulative feature, it was replaced by the following difference by our feature engineering step:

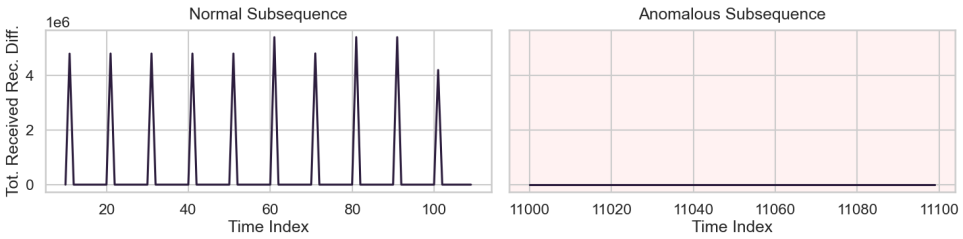


Figure 4.3: Time plot of the difference in total number of received records for a normal and an anomalous subsequence of lengths 100 in the T3 trace `10_3_1000000_75`.

$$v_t^{\text{diff}} := v_t^{\text{original}} - v_{t-1}^{\text{original}}, \tag{4.1}$$

where $v_t^{\text{original}} \in \mathbb{R}_+$ generally refers to a cumulative feature value at time index t in a sequence, and $v_t^{\text{diff}} \in \mathbb{R}$ to its corresponding difference. For our feature, having a value of zero for v_t^{diff} therefore means no *new* records have been received at time index t, which naturally occurs for most time indices in Exathlon's traces. As described in Appendix A.1, every application in Exathlon was indeed run with a fixed *processing period*, which defined the rate at which the application triggered new computations, and thus considered new input records to process. For non-windowed applications, this processing period corresponds to the batch interval, which was always set to five seconds. For windowed applications, it corresponds to the window slide parameter, set to 20 seconds for application 3, and 10 seconds for applications 8, 9 and 10. For traces of application 10, like the one shown in Figure 4.3, a *subsequence* of zeros for the difference in total number of received records is therefore normal if and only if this subsequence spans nine time steps. This is not the case during T3 events, which last 16 minutes (i.e., around 960 time steps) on average.

CPU Contention (T4) Events

Figure 4.1d shows a time plot of a CPU Contention (T4) trace. Throughout data collection, T4 events could have different impacts on a recorded application and features depending on (i) where this application's components were running and (ii) where the CPU-intensive process was launched on the cluster. As such, T4 events can further be categorized into the three following subtypes, listed below in decreasing order of expected difficulty:

- **Crash-Inducing Contention.** Contention that impacted the application so much that it eventually led it to crash (e.g., the fourth anomaly instance in Figure 4.1d).
- **Regular CPU Contention.** Contention that impacted the application in a milder way, without leading it to crash (e.g., the first, second, fifth and sixth anomaly instances in Figure 4.1d).
- **No-Impact Process.** CPU-intensive process that was launched outside the scope of the monitored application's driver and executors, and thus had either no or a very slight impact on its behavior (e.g., the third anomaly instance in Figure 4.1d). Such T4 events only really affected the OS metrics of the node that hosted the CPU-intensive process. Since we removed all such OS metrics from our feature set, **we never consider this subtype** of T4 events in our experiments.

The removal of explicit, CPU-related OS metrics from our feature set made all T4 events more subtle and challenging to detect. As shown in Figure 4.2b, T4 events that had some impact on the applications could induce point, contextual or no anomalies for the processing delay. Most of the "easier" anomalies correspond to crash-inducing

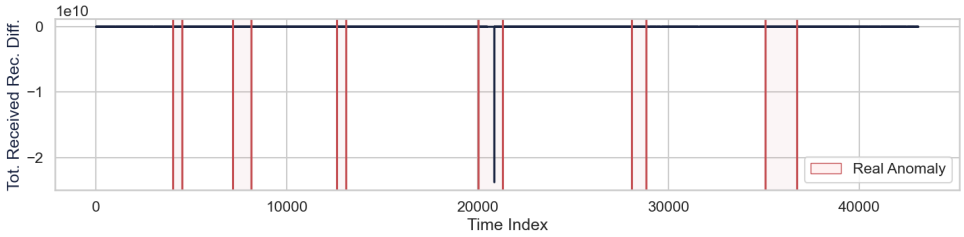


Figure 4.4: Time plot of the difference in total number of received records in the T4 trace `9_4_1000000_78`. The fourth anomaly instance induced an application crash.

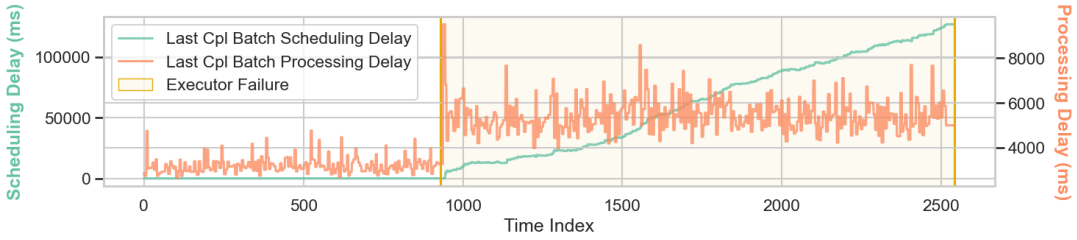
contentions, both because of their stronger impact on the applications and because of the crashes themselves, which had the effect of resetting all the counter features. From Equation 4.1, such resets made the corresponding differences go from their usual, moderate positive values to very large negative numbers (i.e., $v_t^{\text{diff}} := 0 -$ "large original count"). We illustrate this in Figure 4.4, showing a time plot of the difference in total number of received records for the T4 trace `9_4_1000000_78`, where the fourth anomaly instance induced an application crash.

Driver Failure (T5) and Executor Failure (T6) Events

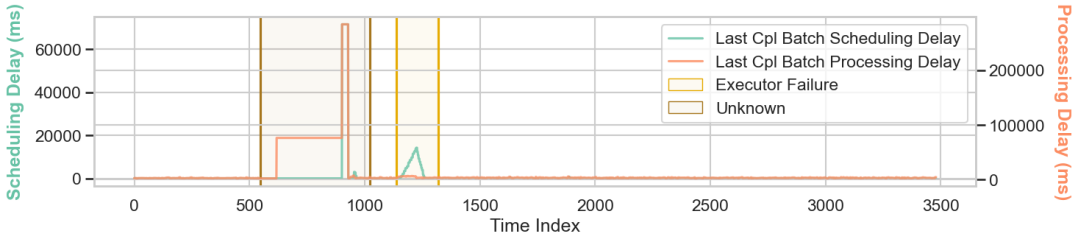
Figure 4.1e shows a time plot of a Process Failure (T5) trace, containing both Driver Failure (T5) and Executor Failure (T6) events. T5 events are very short and point-based in nature. Unlike crash-inducing CPU contentions, their purpose and immediate effect were to lead to an application crash. As such, the significant point anomalies induced by crashes through counter resets (described earlier for crash-inducing CPU contentions) took a greater part of T5 ranges, making them more globally point-based than the anomalies induced by other event types.

Our feature set containing executor features "averaged across active executors", and not for each executor separately, made most T6 events more subtle and challenging to detect. Indeed, an executor failing can now only be detected through the impact this failure had either on other application features, or on "active executor averages" during the (short) period the failed executor was being replaced by a new one. All T6 events involved a single Spark executor at a time, and could belong to one of three subtypes listed below by decreasing order of expected difficulty:

- **Failure without Replacement, with Buildup in Delay.** Failed executor that was not replaced by another, which induced a buildup in scheduling delay ending either with an application crash or the end of recording. An example of this subtype is the executor failure of trace `2_5_1000000_88`, shown in Figure 4.5a.
- **Failure with Replacement, with Temporary Delay.** Failed executor that was shortly replaced by another, which induced an increase in scheduling delay during the replacement period. An example of this subtype is the T6 event of trace `1_5_1000000_86`, shown in Figure 4.5b.
- **Failure without Replacement, with Temporary Delay.** Failed executor that was not replaced by another, which induced a short, temporary increase in scheduling delay (meaning that after a short adjustment period, the application could run properly with one less executor). An example of this subtype is the T6 event of trace `9_5_1000000_84`, shown in Figure 4.1e.



(a) Failure "without replacement, with buildup in delay" in trace 2_5_1000000_88.



(b) Failure "with replacement, with temporary delay" in trace 1_5_1000000_86.

Figure 4.5: Example executor failure instances projected on the last completed batch scheduling and processing delays.

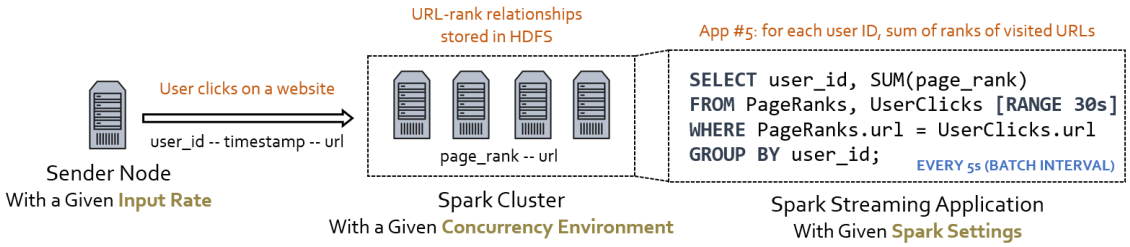
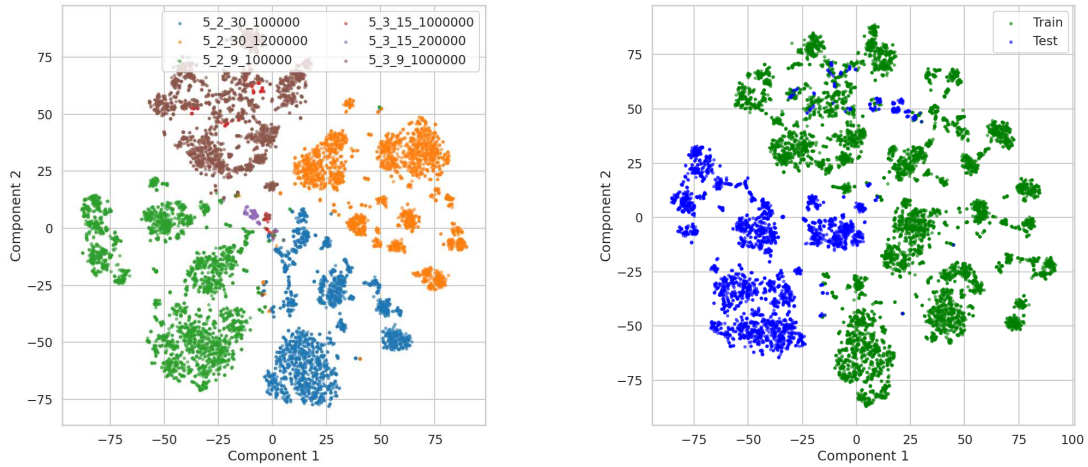


Figure 4.6: The diversity in Spark settings, data sender input rate and concurrency environment induce diverse normal behaviors even within runs of a same application.

4.3.2 Diversity and Shift in Normal Behaviors

The Exathlon benchmark was built to reflect a set of key challenges typically encountered in AIOps settings. Among these challenges, and as hinted by the contextual nature of most anomaly types, are the *diversity and shift in normal behaviors* for the recorded application runs. The first reason for this diversity is the presence eight different Spark Streaming applications (i.e., *entities* in AIOps terms [9]) in both the training and test set of our experimental setup. Even for a same Spark Streaming application, however, different *runs* could be launched and recorded in vastly different *contexts*, mainly characterized by the three following high-level factors (also illustrated in Figure 4.6):

- The **Spark settings** set for the application run. These settings included the *processing period* described earlier (i.e., the batch interval or the window slide), which was always constant across runs of a same application in Exathlon. Settings that could vary across all runs were however (i) the number of active Spark executors and (ii) the "memory profile" (including the maximum memory set for the driver block manager, executors JVM and garbage collection, as further detailed in Appendix A.1). Both these aspects had either a direct or indirect impact on the baselines values taken by a lot of features (e.g., the driver block manager memory usage).



(a) Data records colored by context.

(b) Data records colored by dataset.

Figure 4.7: t-SNE scatter plots of application 2's normal data, undersampled to 10,000 data records balanced by context.

- The **input rate** of the data sender, which defined the (approximate) rate at which data records were sent to the application. This input rate had a direct effect on the *volume* of data periodically processed by an application run, and therefore on a lot of baseline values for its recorded features (e.g., the last completed batch processing delay).
- The **concurrency environment** of the cluster, defined as the set of other programs that were running on the four nodes during the recording of a trace. Thanks to the resource isolation performed by Spark, this factor only mainly impacted the OS metrics recorded for the nodes by the `nmon` software. Because we removed such metrics from our feature set, **we choose to ignore this concurrency factor** in this study.

All in all, we consider the "normal behavior" in a trace to be mainly determined by its **trace characteristics**, defined as the combination of its **entity** and **context**. In Exathlon and our setup, the entity corresponds to the Spark Streaming application recorded, and the context to the Spark settings and input rate used for the application run.

Using this terminology, the *diversity* in normal behaviors stems from the diversity in trace characteristics in our experimental setup, while the *shift* in normal behavior from the training to the test data stems from the shift in context for each of the eight entities. The illustrate these challenges, Figure 4.7 shows t-SNE scatter plots [116] of application 2's *normal* data, undersampled to 10,000 data records balanced by context. The *diversity* challenge is shown in Figure 4.7a, where data records are colored by context (with context labels indicating the processing period, number of Spark executors, maximum executors memory and data sender input rate, respectively). As we can see on this plot, different contexts appear as distinct clusters, constituting a *multimodal* distribution for data records. Figure 4.7b illustrates the *shift* challenge, with the same data records colored this time by dataset in our experimental setup. As we can see, the different contexts induce a distribution shift from the training to the test data, even within normal records of a same application. This distribution shift is the reason for the contextual nature of most of the anomalies in Exathlon's dataset.

4.4 Compared Methods and Hyperparameters

This section presents the unsupervised methods and hyperparameters considered in our study. We start by describing our general model training and selection procedure for all the deep learning methods. We then present the methods in turn, recalling their categories from Schmidl et al. [19], and separating those that perform a **point modeling** of the training data (i.e., consider windows of length $L = 1$) from those that perform a **sequence modeling** (i.e., consider windows of length $L > 1$, set to 20 in our experiments).

4.4.1 Model Training and Selection for Deep Learning Methods

All of the deep learning methods considered used the same random 20% of training data as validation, sampled in a stratified manner with traces as strata. For each undisturbed trace, a contiguous period of 20% of records was randomly selected. For each disturbed trace, the validation period was fixed so as to obtain a similar distribution of event types in training and validation data (with the exception of T2 traces, which were all used as training because of their shorter lengths). We adopted this strategy in order to use the same training, validation and test data in our unsupervised and weakly-supervised scenarios, with training and validation anomalies simply being removed in the unsupervised case.

Unless mentioned otherwise, all deep learning methods were trained with a Stochastic Gradient Descent (SGD) strategy, using mini-batches of size B, the AdamW optimizer [117], and a weight decay coefficient of 0.01. For all methods and sets of hyperparameters, we considered a grid of learning rate values $\eta \in \{1e{-}5, 3e{-}5, 1e{-}4, 3e{-}4\}$, and selected the learning rate that yielded the lowest validation loss (i.e., the best *modeling* performance, like in [32]). All methods were trained for 300 epochs by default, using early stopping and checkpointing on the validation loss with a patience of 100 epochs.

4.4.2 Point Modeling Methods

Point modeling methods model individual data records ($L = 1$), assumed independent and identically distributed (i.i.d). As such, they only rely on our feature engineering and anomaly score smoothing to capture the sequential aspect of the data. We include the following point modeling methods in our study:

- Isolation forest [58] (**iForest**) as an isolation tree method.
- Principal Component Analysis (**PCA**) [56] and Dense Autoencoder (**Dense AE**) [52, 53] as reconstruction methods.
- Dense Deep SVDD [54] (**Dense DSVDD**) as an encoding method.
- The Mahalanobis method [56, 51] (**Maha**) and Dense Variational Autoencoder (**Dense VAE**) [57] as distribution methods.

Isolation forest trains an ensemble of trees to isolate the samples in the training data, and defines the anomaly score of a test instance based on the average path length required to reach it using the trees. We report its performance with the following hyperparameters (using the default values of Scikit-Learn [108] 1.0.2 for the ones not mentioned):

- A number of trees in $\{50, 100, 200, 500, 1000\}$.
- A maximum number of samples used by each tree in $\{256, 512, 2048, 8192, 32768\}$.
- A maximum number of features used by each tree of 64.

As reconstruction methods, PCA and Dense AE define anomaly scores of test vectors as their mean squared reconstruction errors from a transformed (latent) space. The transformation of PCA is a projection on the linear hyperplane formed by the principal components

of the data. We report its performance with the following preprocessing and hyperparameters (using the default values of Scikit-Learn 1.0.2 for the ones not mentioned):

- A standardization of the input samples.
- A number of principal components (latent dimension) in $\{16, 64, 128, 95\%, 99\%, M\}$, where 95% and 99% correspond to the latent dimension preserving 95% and 99% of the training data variance, respectively, and $M = 237$ is our input dimensionality after feature engineering.

The transformation of the Autoencoder method is a non-linear mapping to a latent encoding learned by a neural network that was trained to reconstruct input data from it. With Dense AE, we consider a fully-connected architecture for this neural network, and report its performance with the following preprocessing and hyperparameters:

- A standardization of the input samples.
- A single hidden layer of 200 units for both the encoder and the decoder.
- The Rectified Linear Unit (ReLU) activation function for all the layers except the output, for which we do not use any activation function.
- An encoding dimension in $\{16, 64\}$.
- A mini-batch size $B = 32$.

The Mahalanobis and VAE methods define anomaly scores of data samples as their deviation from an estimated data distribution. The Mahalanobis method estimates this distribution as a multivariate Gaussian, and defines the anomaly score of a test vector as its squared Mahalanobis distance from it. As such, it does not require any hyperparameters, and we therefore report its performance using only a standardization of the input samples. Dense VAE estimates the data distribution using a fully-connected variational autoencoder, with the anomaly score of a test point derived by drawing multiple samples from the probabilistic encoder, and averaging the negative log-likelihood of the reconstructions obtained from each of these samples. We report its performance using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- A single hidden layer of 200 units for both the encoder and the decoder.
- An encoding dimension in $\{16, 64\}$.
- The ReLU activation function for all the layers except the encoding and output. To improve numerical stability, we adopt a similar strategy to Xu et al. [103], and derive the standard deviations of encodings and outputs using softplus activations shifted by a small constant ϵ set to 1e$-$4.
- A mini-batch size $B = 32$.
- A number of samples drawn of 256 to derive the anomaly score of a test example.

The Dense DSVDD method trains a fully-connected neural network to map the input data to a latent representation enclosed in a small hypersphere, and then defines anomaly scores of test samples as their squared distance from this hypershere's centroid. We use the implementation of Ruff et al. [62] with the "One-Class Deep SVDD" objective (assuming most of the training data is normal) and their initialization of the encoder weights from a pretrained autoencoder model. We report the performance of Dense DVSDD with the following preprocessing and hyperparameters (using the same values as the original implementation for those not mentioned):

- A standardization of the input samples (we also tried the original paper's strategy of normalizing the inputs and using a sigmoid activation function for the output layer, but this did not lead to a better performance).

- A single hidden layer of 200 units for the encoder and the decoder (with the decoder only being used for pretraining).
- An encoding dimension in $\{16, 64\}$.
- The Leaky ReLU activation function with a negative slope coefficient $\alpha = 0.01$ for all the layers except the encoding and decoder output (like in the original implementation).
- A mini-batch size $B = 200$ (like in the original implementation).
- A pretraining phase of 150 epochs, followed by a training phase of 150 epochs (which makes the same total number of 300 epochs as the other methods).
- The same learning rate and optimization strategy for the pretraining and training phases.
- The same grid of learning rate values as the other methods, but dividing the learning rate by 10 after 50 epochs (like in the original implementation). Larger learning rate values were also tried due to this scheduling, but did not produce better results.
- A weight decay coefficient of 1e−6 (like in the original implementation).

4.4.3 Sequence Modeling Methods

Sequence modeling methods model wider *windows* of data records ($L = 20$ here), which offers them the capacity of explicitly considering the temporal aspect of the data. We include the following sequence modeling methods in our study:

- Recurrent Autoencoder [52, 53] (**Rec AE**) and **TranAD** [14] as reconstruction methods.
- **LSTM-AD** [49] as a forecasting method.
- Recurrent Deep SVDD [54] (**Rec DSVDD**) as an encoding method. We also tried to include the more recent encoding method DCDetector [55], but did not retain it due to its poor performance on our dataset.
- Recurrent VAE (**Rec VAE**) as a distribution method.

Rec AE uses the same modeling and scoring strategy as Dense AE, with the fully-connected neural network architecture replaced by a recurrent one. Figure 4.8 illustrates the general form we adopted for our recurrent autoencoders, including 1D convolutional and recurrent layers. In this design, the encoder first consists of an optional stack of 1D convolutional layers, which in this example contains a single layer labeled `Conv1D`$(32, 5, s)$, to indicate it has 32 filters of size 5 and a stride length hyperparameter s. These layers result in a new latent window length $L' \leq L$ for an input window, with one feature map per filter in the last layer. These feature maps get sent to an optional stack of GRU layers, here shown as a single layer labeled `GRU`$(64, \text{Last})$, to indicate it has 64 units and returns its outputs for the last time step only. These layers are followed by a fully-connected layer that outputs the final encoding. This encoding is provided as input to the decoder, which repeats it L' times to match the window length of the data after the 1D convolutions. This repeated vector goes through a stack of GRU layers typically defined symmetrically to the encoder's, except it now returns its outputs for each of the L' time steps. These outputs finally get passed to a stack of 1D transposed convolutional layers defined symmetrically to the encoder's 1D convolutional layer stack, except for the output layer using M filters to match the input dimensionality. We report the performance of Rec AE using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- An encoder with a 1D convolutional layer using 32 filters of size 5, a stride length of 1 and the ReLU activation function, followed by a GRU layer of 64 units using the

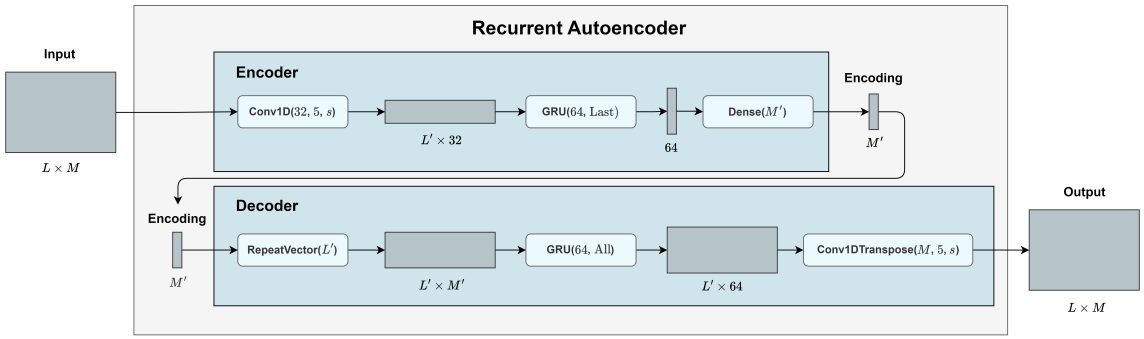


Figure 4.8: General form of our recurrent autoencoder architectures.

hyperbolic tangent (tanh) activation function, and a fully-connected layer to output the encoding.
- A decoder defined symmetrically to the encoder as per the design of Figure 4.8.
- An encoding dimension in $\{64, 128\}$, with the ReLU activation function for the encoding layer.
- A mini-batch size $B = 32$.

TranAD uses a transformer-based model with self-conditioning, an adversarial training procedure and model-agnostic meta learning (MAML). It relies on two encoder-decoder networks, with the first encoder considering the current input window, and the second one considering a larger *context* of past data in the window's sequence. The method defines the anomaly score of an input window as the average of its reconstruction errors coming from two decoders and inference phases, with the second phase using the reconstruction error from the first phase as a focus score to detect anomalies at a finer level. Compared to the other methods considered, TranAD therefore considers training windows augmented with their past sequence data, which prevented us from applying the simple window balancing strategy described in Section 4.2.5. We report the performance of TranAD using the implementation of Tuli et al. [14], only specifying manually the following preprocessing and hyperparameters:

- A normalization of the input samples (like in the original implementation). We also tried the strategy of standardizing the inputs and using no activation function for the output layer (like the other methods), but this did not lead to a better performance.
- A number of encoder hidden units in $\{64, 128\}$.
- The same grid of learning rate values as the other methods, but multiplying the learning rate by 0.9 every 5 epochs (like in the original implementation). Larger learning rate values were also tried due to this scheduling, but did not produce better results.
- A mini-batch size $B = 128$ (like in the original implementation).
- A weight decay coefficient of 1e$-$5 (like in the original implementation).

LSTM-AD trains a stacked LSTM network to predict the next l data records from the first $L-l$ of a window. Originally designed for univariate time series, this method produces an l-dimensional vector of forecasting errors for each data record in a test sequence, with one component for each position held by this record in forecast windows of length l. The method then fits a multivariate Gaussian distribution to the error vectors it produced in a validation set, and defines the anomaly score of a test record as the negative log-likelihood of its error with respect to this distribution. In this work, we adapt LSTM-AD to multivariate data by considering the $l \times M$ matrix of forecasting errors made

for a data record at each time step and feature, and averaging these errors across the feature dimension to get the l-dimensional vector of the original method. We report the performance of LSTM-AD using the following hyperparameters:

- A standardization of the input samples.
- A forecast window length $l = 10$ (i.e., half of our window length $L = 20$).
- A version with a single LSTM layer of 128 units, and a version with two LSTM layers of 128 units each, all using the tanh activation function.
- A mini-batch size $B = 32$.

Rec VAE uses the same modeling and scoring strategy as Dense VAE, with the fully-connected neural network architecture replaced by a recurrent one following the design of Figure 4.8. We report the performance of Rec VAE using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- An encoder with a 1D convolutional layer using 32 filters of size 5, a stride length of 1 and the ReLU activation function, followed by a GRU layer of 64 units using the tanh activation function, and a fully-connected layer to output the encoding parameters.
- A decoder defined symmetrically to the encoder as per the design of Figure 4.8.
- An encoding dimension in $\{64, 128\}$.
- The same strategy as Dense VAE for deriving the encoding and output standard deviations.
- A mini-batch size $B = 32$.
- A number of samples drawn of 256 to derive the anomaly score of a test example.

Rec DSVDD uses the same modeling and scoring strategy as Dense DSVDD, with the fully-connected neural network architecture replaced by a recurrent one, adapting the implementation of Ruff et al. [62] to match the design of Figure 4.8. We report the performance of Rec DSVDD with the following preprocessing and hyperparameters (using the same values as the original implementation for those not mentioned):

- A standardization of the input samples.
- An encoder with a 1D convolutional layer using 32 filters of size 5, a stride length of 1, batch normalization and the Leaky ReLU activation function (with a negative slope coefficient $\alpha = 0.01$), followed by a GRU layer of 64 units and a fully-connected layer to output the encoding.
- A pretraining decoder defined symmetrically to the encoder as per the design of Figure 4.8.
- An encoding dimension in $\{64, 128\}$, with no activation function for the encoding layer.
- A mini-batch size $B = 200$ (like in the original implementation).
- A pretraining phase of 150 epochs, followed by a training phase of 150 epochs.
- The same learning rate and optimization strategy for the pretraining and training phases.
- The same grid of learning rate values as the other methods, but dividing the learning rate by 10 after 50 epochs (like in the original implementation).
- A weight decay coefficient of 1e−6 (like in the original implementation).

4.5 Results and Analyses

This section presents and analyzes the results obtained with the unsupervised methods introduced in Section 4.4, using the evaluation strategy of Section 4.2.6. We start by

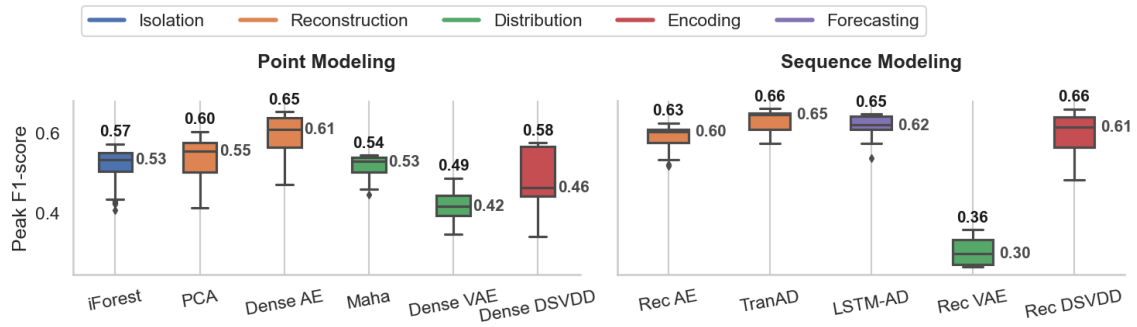


Figure 4.9: Box plots of peak F1-scores achieved by each unsupervised method, separated by modeling strategy (point vs. sequence) and colored by method category.

describing the format in which the results are presented. We then derive our 15 conclusions, labeled **C1-15**, regarding the (i) difficulty of detecting the different event types in test data, (ii) performance within and across point and sequence modeling methods, and (iii) limitations of the best-performing methods in addressing the dataset's challenges.

4.5.1 Format of Results

Figure 4.9 shows the box plots of the peak F1-scores achieved by each unsupervised method across their hyperparameter values. It separates point from sequence modeling methods in two different subplots with a shared y-axis, with boxes colored based on the method category.

Table 4.1 considers the methods with their best-performing hyperparameter values (i.e., that yielded the maximum peak F1-score in Figure 4.9), and reports the average peak F1-score they obtained *within each test trace* for each type of event. For example, considering three traces `A`, `B` and `C`, with only traces `A` and `C` containing T1 events, we would compute the maximum possible F1-score for the method's anomaly scores in traces `A` and `C` separately, as the harmonic mean of Precision and T1 Recall (e.g., getting 0.65 for trace `A` and 0.45 for trace `C`). We would then report the average of these two F1-scores as the method's `Tr-T1` value in Table 4.1 (0.55 in our example). We consider such metrics to study the ability of each method to detect different types of events, when partly factoring out the challenge of the normal behaviors shift across traces.

4.5.2 Difficulty of Event Types

Considering the `Average` row of Table 4.1, the **easiest types** of events to detect across the methods appear to be **T1** (Bursty Input), **T2** (Bursty Input Until Crash) and **T5** (Driver Failure) (**C1**). This observation is consistent with our discussion of Section 4.3, indicating that T1 and T2 events induce contextual anomalies for a lot of features within a trace, with T2 events being "easier" versions of T1 events. As such, we expect most of the poor performance for these types of events to be due to the distribution shift across traces, whose effects were partly factored out in Table 4.1. Obtaining a good performance for T5 events throughout the methods was also expected, as those events lead to pronounced point anomalies for the counter features. In addition to the high average performance, we can see the top-three F1-scores achieved for these event types (shown in **bold** in Table 4.1) are almost perfect. This reflects an ability of the corresponding methods to perfectly separate these types of events from normal data within the test traces.

The **hardest types** of events to detect appear to be **T3** (Stalled Input), **T4** (CPU Contention) and **T6** (Executor Failure) (**C2**). This is again consistent with our discussion

Method	Tr-T1	Tr-T2	Tr-T3	Tr-T4	Tr-T5	Tr-T6
iForest [58]	0.81	0.95	0.29	0.54	0.68	0.45
PCA [56]	0.79	**0.96**	0.23	0.68	0.84	0.49
Dense AE [52, 53]	0.92	0.96	0.67	**0.72**	0.93	**0.54**
Maha [56, 51]	0.82	0.93	0.39	0.57	**0.97**	0.48
Dense VAE [57]	0.51	0.89	0.61	0.55	0.87	0.43
Dense DSVDD [54]	**0.97**	0.96	0.58	0.46	0.91	**0.52**
Point Average	0.80	0.94	**0.46**	0.59	**0.87**	0.49
Rec AE [52, 53]	0.89	**0.97**	0.43	**0.77**	**0.95**	0.51
TranAD [14]	**0.97**	**0.99**	**0.78**	0.51	**0.97**	0.48
LSTM-AD[49]	**0.95**	0.90	**0.77**	0.63	0.67	0.51
Rec VAE [57]	0.36	0.74	**0.80**	0.38	0.59	0.44
Rec DSVDD [54]	0.79	0.99	0.55	**0.72**	0.90	**0.65**
Sequence Average	0.79	0.92	**0.67**	0.60	**0.82**	0.52
Average	**0.80**	**0.93**	**0.58**	**0.60**	**0.86**	**0.51**

Table 4.1: Peak F1-scores achieved by the best-performing unsupervised methods for each event type within a trace (averaged across test traces), with the top-three F1-scores for each event type shown in **bold**.

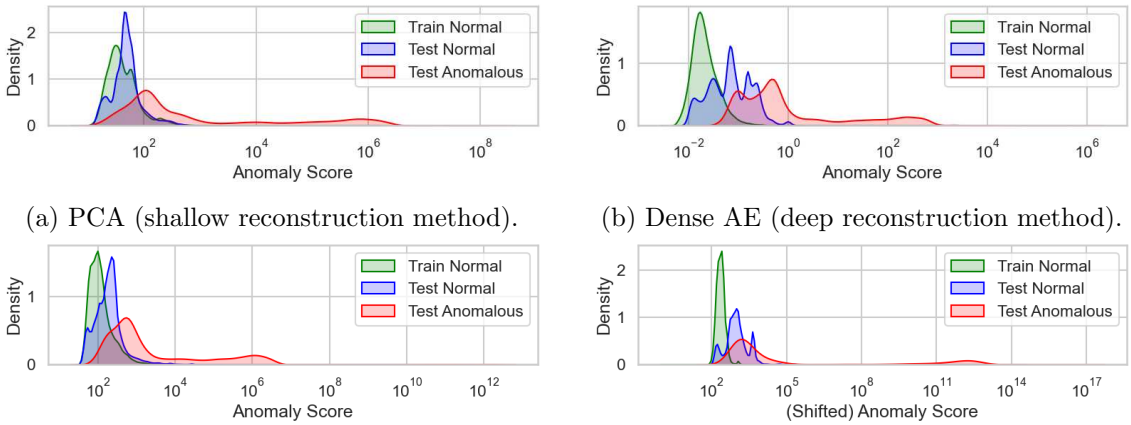
of Section 4.3, with T3 events being more "subtle" than T1 events (in that they do not induce any delays for the application), as well as T4 and T6 including subtypes with even more subtle impacts on the recorded features. Looking at the top-three F1-scores for each type, we can see that T6 events were by far the most challenging type to detect for our methods (even within a given trace). A likely reason of this poorer performance is the averaging of "active executor" features we performed in our feature engineering, which made executor failures far more subtle to detect given our feature set. However, the maximum F1-score of 0.65 achieved by Rec DSVDD seems to indicate that a better detection of T6 events can still be achieved through modeling only, without altering our feature engineering procedure.

4.5.3 Point Modeling Methods

Looking back at the performance across all test traces in Figure 4.9, and focusing on *point modeling* methods only, we can see that the **reconstruction** methods performed the best, while the **distribution** methods performed the worst (**C3**). Among reconstruction methods, the deep method (Dense AE) outperformed the shallow one (PCA), while the shallow method (Maha) outperformed the deep one (Dense VAE) among distribution methods (**C4**).

These two observations can be linked to the distribution shift challenge described in Section 4.3. To illustrate this, we show in Figure 4.10 the KDE plots of the anomaly scores assigned by the point modeling reconstruction and distribution methods to the training normal, test normal and test anomalous records[1]. On these plots, the separation between the anomaly scores assigned to the test normal and test anomalous records (i.e., between the **blue** and **red** KDEs) directly relates to anomaly detection performance of a method, while the overlap between the scores of training normal and test normal records (i.e., **green** and **blue** KDEs) reflects its "robustness" to the distribution shift from training to test data.

[1]The x-axes being shown in log scale, the anomaly scores of the methods that produced negative values were further shifted so as to get a minimum value of 1 before applying the logarithm function.



(a) PCA (shallow reconstruction method).

(b) Dense AE (deep reconstruction method).

(c) Maha (shallow distribution method).

(d) Dense VAE (deep distribution method).

Figure 4.10: Kernel Density Estimate (KDE) plots of the anomaly scores assigned by reconstruction and distribution methods to training normal, test normal and test anomalous records.

Comparing Figures 4.10c to 4.10a and 4.10d to 4.10b, respectively, we can see that distribution methods, by modeling the training distribution more *explicitly*, tend to produce more similar anomaly scores across the training normal records (i.e., tighter **green** KDEs). However, this tighter modeling of the training distribution also makes these methods more sensitive to the distribution shift from training to test data. This makes them view test normal and test anomalous records as "similarly anomalous" (i.e., high overlap between the **blue** and **red** KDEs), which hinders their test anomaly detection performance.

Comparing now Figures 4.10b to 4.10a and 4.10d to 4.10c, respectively, we can see that the deep methods achieved a better separation between the training normal and test anomalous records (**green** and **red** KDEs) than the shallow methods, but also between the training normal and test normal records (**green** and **blue** KDEs). Given our large data cardinality, the deep methods can indeed model the training data at a finer level than the shallow ones. For distribution methods, the benefit of better separating anomalies from the *training* normal records (**red** vs. **green**) is however counterbalanced by an increased sensitivity to the distribution shift, and thus worse separation of anomalies from the *test* normal records (**red** vs. **blue**).

Finally, considering the point modeling methods in Table 4.1, we can see that all the deep learning methods significantly outperformed all the shallow methods for T3 (Stalled Input) events (**C5**). As described in Section 4.3, a key property of Exathlon's dataset (and our feature engineering) is that a single instance of anomalous event could typically induce a variety of anomaly types for the features, making its information *redundant* in different forms throughout our feature set. This redundancy is especially prevalent for T3 events, which are reflected as point, contextual and collective anomalies at once. From Table 4.1, it appears that the higher capacity of the deep methods can help them discover (some) non-collective anomaly signals from the features more easily than the shallow methods, which makes them perform decently for T3 events despite using a point modeling strategy.

4.5.4 Sequence Modeling Methods

Considering sequence modeling methods in Figure 4.9, we can see that the **distribution** method (Rec VAE) is again the worst-performing one (**C6**). This can be explained similarly to **C3**, with Rec VAE separating well the test anomalies from the normal *training* data (almost perfect **red** vs. **green** KDE separation in Figure 4.11), but also deeming

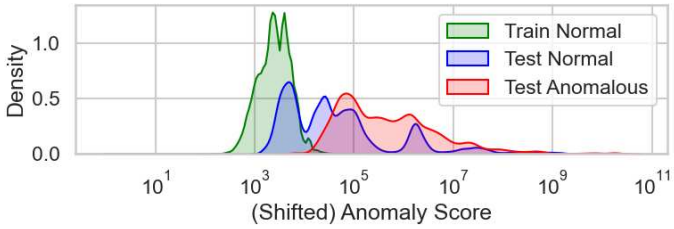


Figure 4.11: Kernel Density Estimate (KDE) plots of the anomaly scores assigned by Rec VAE to training normal, test normal and test anomalous records.

anomalous a large part of the shifting *test* records (poor **red** vs. **blue** KDE separation in Figure 4.11).

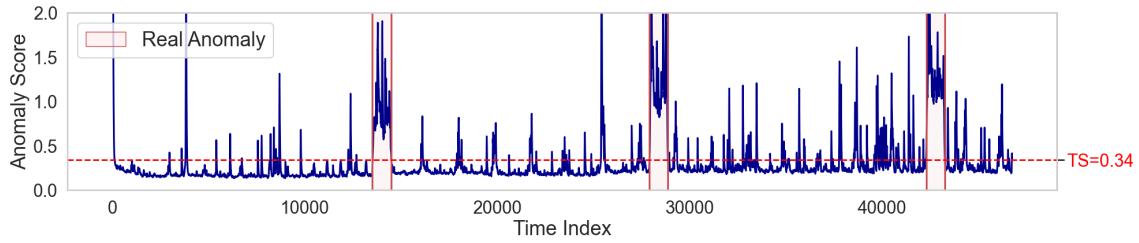
Apart from a vulnerability to the distribution shift challenge of Exathlon's dataset, the Precision of a method can also be hindered by its failure to integrate *noise* into its learned normal behavior. Figure 4.12 shows time plots of the anomaly scores produced by the Rec AE, Rec DSVDD, LSTM-AD and TranAD methods for the records in trace `2_1_100000_60`, highlighting their peak F1-score thresholds and the ground-truth anomaly ranges. From this figure, we can see that LSTM-AD and TranAD were the most robust to the "noisy", minority patterns present in test data, producing much less anomaly scores above their threshold value outside the real anomaly ranges (**C7**). LSTM-AD being robust to noise is consistent with the study of Schmidl et al. [19], and could be explained by this method assigning anomaly scores from a distribution of forecasting error vectors rather than from forecasting errors directly. This result is also expected for the TranAD method, which, by using a larger context of past data windows than the other methods to derive its anomaly scores, tends to be more robust to the less-sustained deviations from normality represented by noise.

4.5.5 Point vs. Sequence Modeling

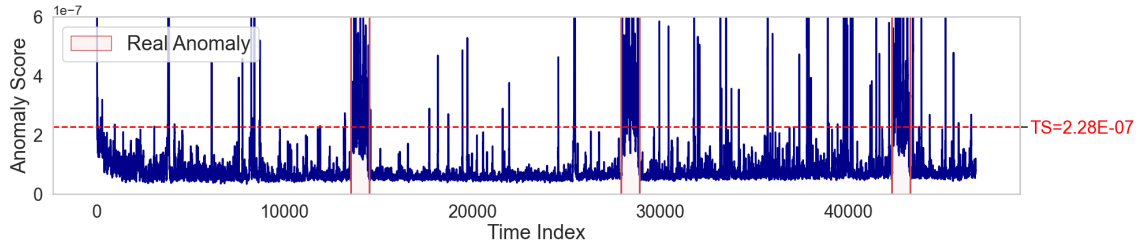
Considering Figure 4.9 again, we can see that both the best "maximum" and "median" performances were achieved by sequence modeling methods (**C8**), with maximum F1-scores of 0.66, 0.66 and 0.65 for TranAD, Rec DSVDD and LSTM-AD, respectively, and median F1-scores of 0.65, 0.62 and 0.61 for TranAD, LSTM-AD and Rec DSVDD, respectively. This advantage over point modeling methods is significant except for the Dense AE method, which could achieve a maximum F1-score of 0.65 and median F1-score of 0.61. This smaller difference with Dense AE can partly be explained by the *redundancy* property of our dataset and feature engineering described in Section 4.5.3, making it possible to detect a large part of events using point modeling only, provided the method has enough capacity to find deviating *point* or *contextual* anomalies in the features. As shown in Figure 4.9, however, although detecting such events is partly *possible* with point modeling methods, it is in general *harder* to achieve than with sequence modeling methods, which perform better on average (**C9**).

As shown in Table 4.1, the better performance of sequence modeling methods was mainly due to a significantly better detection of **T3** (Stalled Input) events within traces (with an average F1-score of 0.67 across the sequence modeling methods vs. 0.46 across the point modeling methods) (**C10**). This is consistent with our description of Section 4.3, and T3 events tending to induce more *collective* anomalous patterns in the features than the other event types.

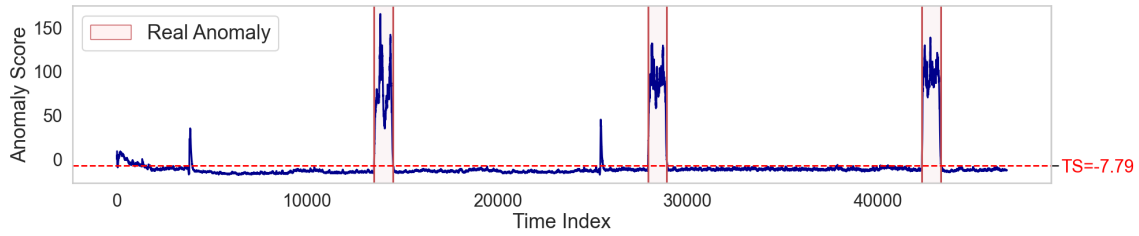
Still from Table 4.1, we can see that sequence modeling methods however performed worse than the point modeling ones for **T5** (Driver Failure) events within traces (with an



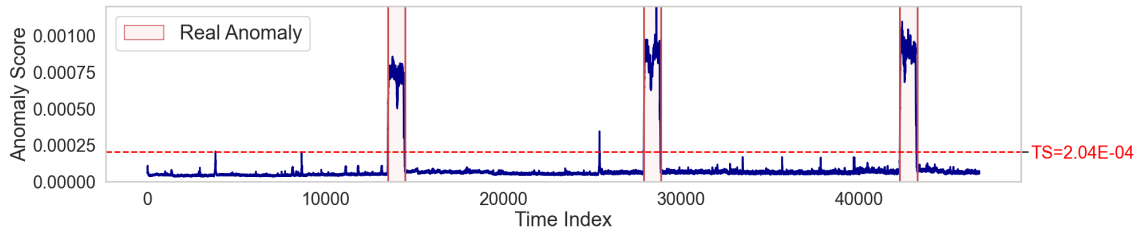
(a) Anomaly scores of Rec AE.



(b) Anomaly scores of Rec DSVDD.



(c) Anomaly scores of LSTM-AD.



(d) Anomaly scores of TranAD.

Figure 4.12: Time plots of the anomaly scores of Rec AE, Rec DSVDD, LSTM-AD and TranAD for the records in trace `2_1_100000_60` (Bursty Input), highlighting their peak F1-score thresholds and the ground-truth anomaly ranges.

average F1-score of 0.82 across the sequence modeling methods vs. 0.87 across the point modeling methods) (**C11**). This difference in performance is less significant than for T3 events, but is again relatively expected, since T5 events correspond to the shortest, most "point-based" anomalies in the data. As such, they induce anomalous signals that are easily identifiable using point modeling, but more susceptible of "fading" within the larger contexts of mostly-normal windows used by sequence modeling methods.

4.5.6 Limitations of Best-Performing Methods

As mentioned in Section 4.5.5, the best-performing methods in our study were TranAD, Rec DSVDD and LSTM-AD. In particular, TranAD achieved the highest "maximum" and "median" peak F1-scores (0.66 and 0.65, respectively), as well as a good general separation of the different event types within test traces (with four out of six F1-scores in the top-three for their event type in Table 4.1). This overall performance however remains relatively low, with even the most effective methods showing limitations to address some AIOps challenges of the Exathlon dataset.

The first factor hindering performance relates to the concept of **false negatives** and **Recall**, with methods assigning anomaly scores that are too *small* for some *anomalous* records in test data. This pertains to our discussion of Section 4.5.2 about the "hardest" event types to detect, and can be associated with the AIOps challenge of *identifying anomaly signals for the hardest T4 (CPU Contention) and T6 (Executor Failure) events from a high-dimensional feature set* (**C12**). As we will see in Chapter 6, an effective solution to partly address this challenge is to augment the training data with a few labeled records for these types of events.

The second factor relates to the concept of **false positives** and **Precision**, with methods assigning anomaly scores that are too *large* for some *normal* records in test data. This pertains to many observations from our study, and can be associated with the following AIOps challenges:

- Being resilient to the *shift in normal behaviors* that can occur from the training to the test data (**C13**).
- *Properly integrating "noisy", minority patterns as part of the learned normal behavior* (**C14**). Although less impactful than the first challenge on the final performance, this challenge remains important to address in practice to avoid repeated false alarms by the detection system.

Throughout this study, this first challenge emerged as the main impediment to optimal performance, and will be our main focus in the rest of this work. Figure 4.13 further illustrates it for our best-performing baseline, TranAD, by showing the ridgeline plot of its anomaly scores for the records of each type within the 23 test traces. Specifically, this figure shows a row per test trace, with a given KDE plot corresponding to the distribution of the anomaly scores assigned by the method to the records of a particular type only (i.e., either records within "normal" ranges, or within ranges of a specific type of event from T1 to T6), also highlighting in red its peak F1-score threshold. As we can see from this figure, although TranAD can separate most of the anomalies from the normal data within a given trace, the *misalignment* of the anomaly scores it assigned to *normal* records in *different traces* prevents it from achieving an optimal detection using a *single*, *global threshold*. As an example, we can see that the normal and anomalous records in traces `1_2_100000_68` and `1_5_1000000_86` are quite well separated. However, most of the anomaly scores assigned to the normal records in trace `1_2_100000_68` are (i) above the optimal threshold (which makes them false positives), but also (ii) similar to a lot of anomaly scores assigned to T6 anomalies in trace `1_5_1000000_86`. In other terms, although the model recognizes

that the normal data in trace 1_2_100000_68 is "less abnormal" than the anomalies of this same trace, it also perceives it as "similarly abnormal" to some anomalies in the other context of trace 1_5_1000000_86.

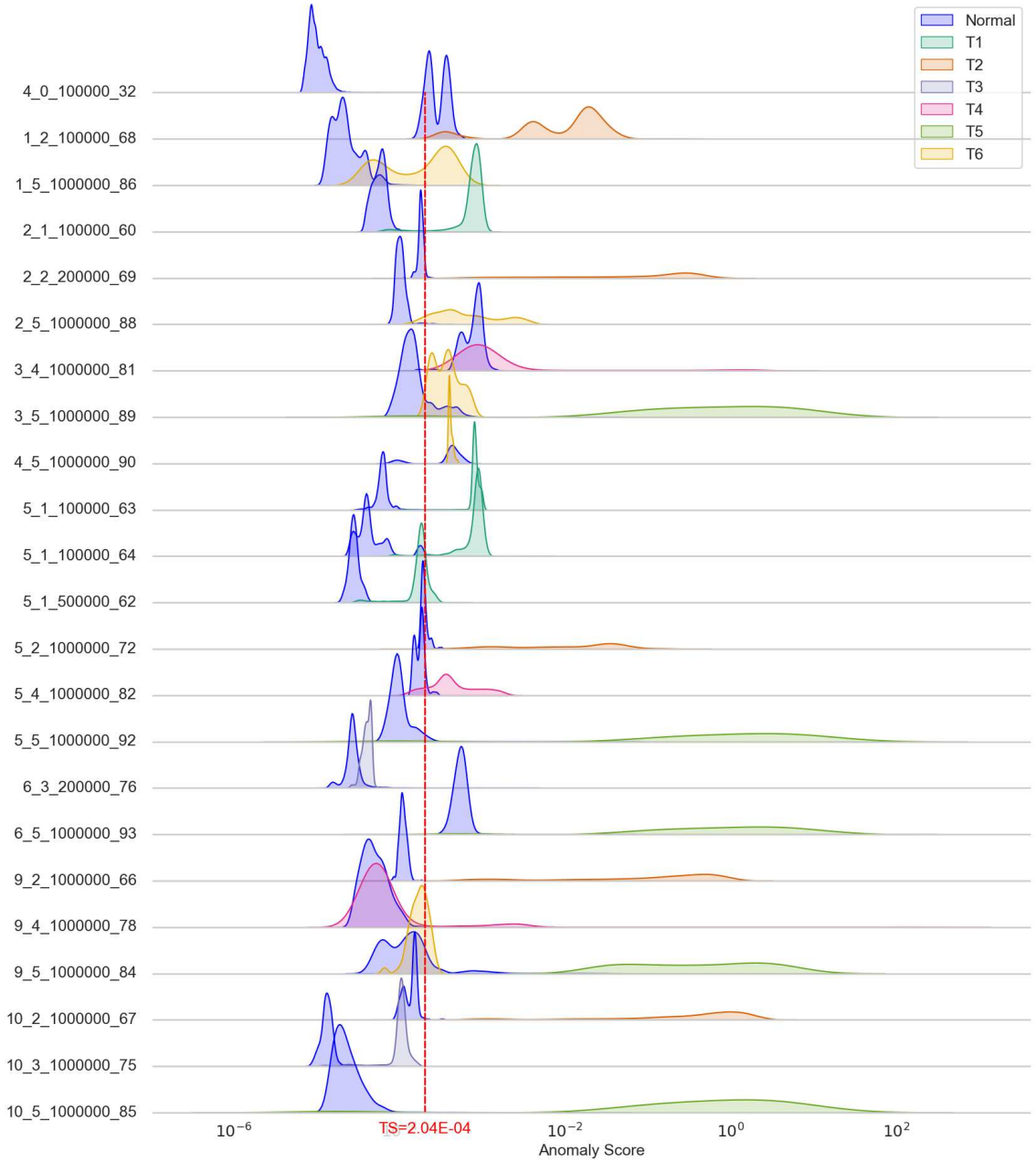


Figure 4.13: Ridgeline plot of TranAD's anomaly scores for the records of each type (i.e., in "normal" and T1 to T6 ranges) in each test trace, with its peak F1-score threshold highlighted in red.

Figure 4.14 shows a similar ridgeline plot for the Rec DSVDD method. As we can see, Rec DSVDD appears more resilient to the normal behavior shift, showing a better alignment for the anomaly scores it assigned to normal records in different test traces. Because the objective of Deep SVDD is to learn a mapping from the input space to a *compact* latent hypersphere, this method tends to *enforce a similar representation of its input samples no matter the context they come from*, which benefits context generalization (**C15**). We will further discuss and leverage this implicit generalization property in Chap-

ter 6, demonstrating its advantage over a less-constrained encoding strategy. We however note that the *implicit* nature of Rec DSVDD's generalization makes it imperfect to address the normal behavior shift challenge. This is for instance illustrated by the lower anomaly scores assigned in traces `5_1_100000_63`, `5_1_100000_64` and `5_1_500000_62`, showing the contexts of these traces were deemed less anomalous than others by Rec DSVDD, which led it to miss T1 events when using its optimal global threshold.

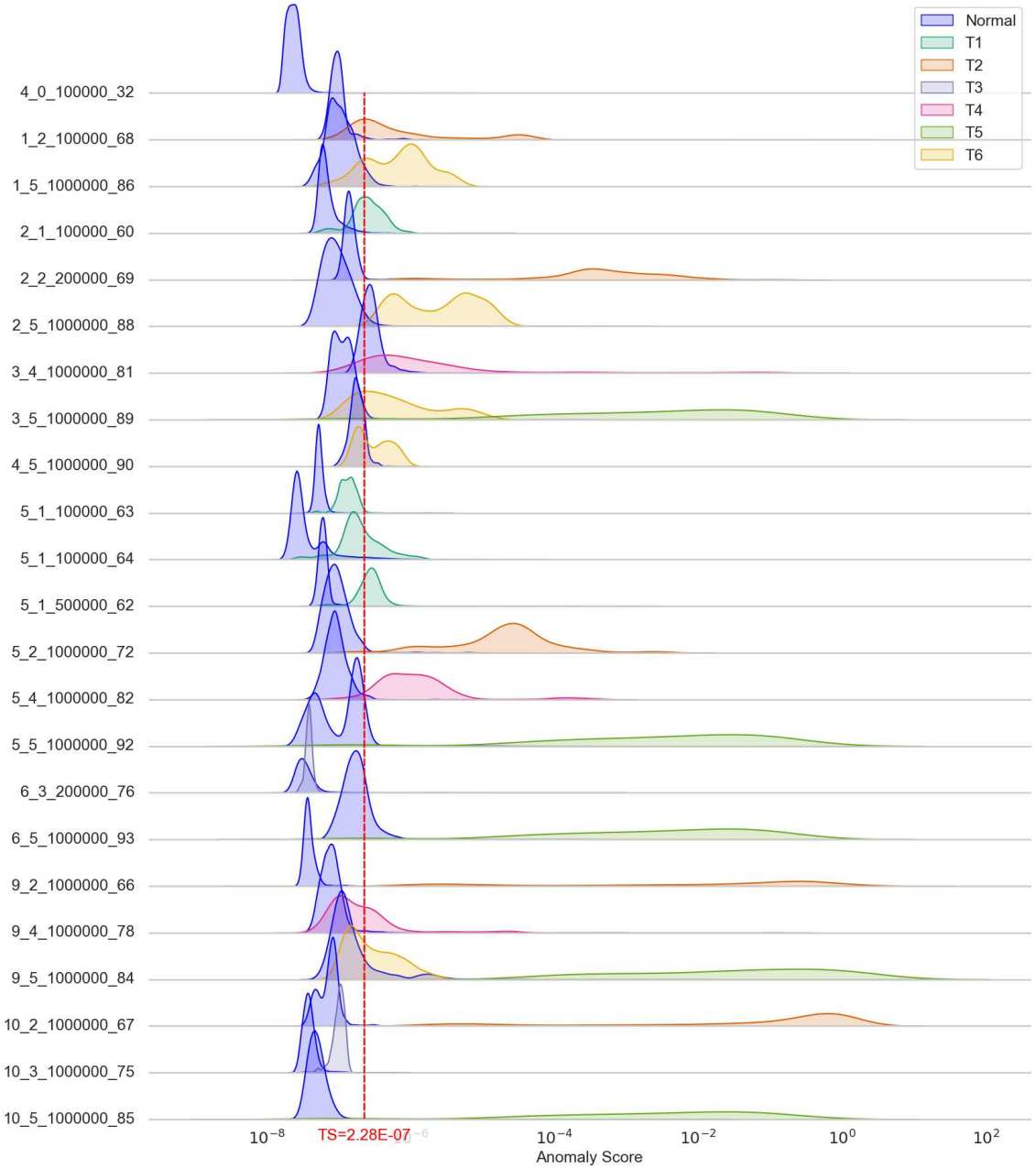


Figure 4.14: Ridgeline plot of Rec DSVDD's anomaly scores for the records of each type (i.e., in "normal" and T1 to T6 ranges) in each test trace, with its peak F1-score threshold highlighted in red.

Limitations of TranAD to address the second challenge are illustrated in Figure 4.15, showing the time plot of its anomaly scores in the (Stalled Input) trace `6_3_200000_76`, whose first half contains a particularly high level of noise. This figure shows that, although TranAD was found more robust to noise than alternative baselines (see **C7**), it

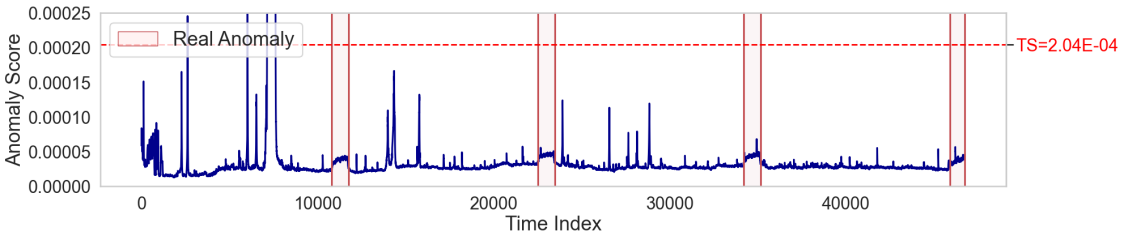


Figure 4.15: Time plot of TranAD's anomaly scores in trace 6_3_200000_76, highlighting its peak F1-score threshold and the ground-truth anomaly ranges.

is still limited in differentiating heavier and more sustained noisy patterns from the relevant anomalies in test data. This is a recurrent shortcoming of expressive unsupervised methods, caused by their lack of prior knowledge about the deviating patterns that are *relevant* to the analysts (the labeled event types in our case), as opposed to other maintained abnormal behaviors (defined as *noise* here, since they simply hinder the analysis). For this reason, our main method for addressing this challenge will be through the weakly-supervised learning framework of Chapter 6.

4.6 Summary and Conclusions

In this chapter, we conducted an experimental study of representative unsupervised anomaly detection methods against the Exathlon benchmark and dataset. We started by formalizing the problem statement considered and describing our experimental setup. We then discussed the characteristics of the Exathlon dataset in more detail, highlighting the relationships between its different event types and the anomaly types commonly defined for time series in the literature. This discussion revealed some key properties of this dataset, such as the prevalence of anomalies that are *contextual* to the **trace characteristics**, conditioning the behavior of data records in a trace and defined as the combination of the recorded **entity** (the Spark Streaming application) and **context** (the Spark settings and data sender input rate used). Using this terminology, we saw that the *variety* and *shift* in normal behaviors observed in our experimental setup could be explained by a *variety in trace characteristics* and a *shift in context*, respectively. Besides this prevalence of contextual anomalies, our discussion also revealed that a same anomalous event could induce an important *redundancy* of the anomaly signal across the features, with *point* and *collective* anomalies also being represented, especially for T3 (Stalled Input) events. After introducing the unsupervised methods compared, we finally presented the results and main analyses of our study, which led to **15 conclusions**, summarized, reorganized and grouped by similarity below:

- **C3, C4, C6, C13, C15.** Reconstruction methods performed the best, while distribution methods performed the worst both within point and sequence modeling methods. The use of deep learning was beneficial among point reconstruction methods, while it degraded performance for point distribution methods. In general, the more a method precisely and explicitly models the training data, the less we can expect it to address our normal behavior shift challenge. This explains why distribution methods performed the worst, and why the coarser density estimation of the shallow distribution method was beneficial. Across all the methods, this AIOps challenge of *normal behavior shift* was identified as the main impediment to optimal performance, with only Deep SVDD addressing it to some extent through the

generalization implicitly induced by its learning objective.

- **C8.** Both the best "maximum" and "median" performances were achieved by sequence modeling methods, showing a significant advantage over all point modeling methods except Dense AE.
- **C1.** Within test traces, the easiest types of events to detect across the methods were T1 (Bursty Input), T2 (Bursty Input Until Crash) and T5 (Driver Failure), due to their significant impact on the features, mainly in the form of contextual anomalies.
- **C2, C12.** Within test traces, the hardest types of events to detect were T3 (Stalled Input), T4 (CPU Contention) and T6 (Executor Failure), due to their more subtle impact on the features. For T4 and T6, this difficulty of detection can be linked to the AIOps challenge of *identifying anomaly signals for the hardest event types from a high-dimensional feature set*.
- **C5.** Within test traces, all deep learning methods significantly outperformed all shallow methods in detecting T3 (Stalled Input) events. This mainly relates to the redundancy of anomaly signals in the Exathlon dataset, with deep learning methods being able to detect T3 events efficiently, be it from collective anomalies or not.
- **C9, C10.** Within test traces, sequence modeling methods outperformed point modeling methods for T3 (Stalled Input) events. Although detecting such events is (partly) possible for point modeling methods (due to the redundancy in anomaly signals), it is in general harder to achieve for them than for sequence modeling methods, which can rely on the significant collective component of T3 events to detect them.
- **C11.** Within test traces, point modeling methods outperformed sequence modeling methods for T5 (Driver Failure) events, due to their shorter and more "point-based" nature than the other event types.
- **C7, C14.** LSTM-AD and TranAD were found to be the most "robust" to the noise encountered in test data, producing much less anomaly scores above their optimal threshold outside the real anomaly ranges. These methods however remain limited in their ability to *differentiate heavier, more sustained noisy patterns from our relevant events*, an AIOps challenge that is important to address in practice for preventing repeated false alarms by the detection system.

In particular, this study revealed three main **limitations** for the unsupervised methods compared, listed below in decreasing order of impact on the AD performance:

- **L1.** *A vulnerability to normal behavior shift from training to test data*, with different *contexts* of *normal* operation for the Spark Streaming applications being considered differently abnormal by the methods.
- **L2.** *A production of false negatives for the hardest anomalies given our large number of features*.
- **L3.** *A production of false positives for normal but "noisy", minority patterns in test data*.

In the next chapter, we will focus on explicitly formalizing and addressing **L1**, consistently identified as the main impediment to optimal performance by this study, relying on the framework of domain generalization.

5 Explicit Domain Generalization

The experimental study conducted in the last chapter consistently identified (**L1**) *the vulnerability to normal behavior shift from training to test data* as the main limitation of the baseline unsupervised methods. In this chapter, we start by formally characterizing its associated challenge, relating it in Section 5.1 to the general concept of **domain shift** [20, 21]. In Section 5.2, we propose Domain-Invariant VAE for Anomaly Detection, or **DIVAD**, a domain generalization method decomposing the observed variable into *domain-specific* and *domain-invariant* encodings, and defining anomalies as samples that deviate from the training distribution of domain-invariant encodings only. We apply this method to our experimental setup in Section 5.3, demonstrating its superiority over the unsupervised baselines to address normal behavior shift and thus detect anomalies. Our experiments also analyze and compare different *variants* of DIVAD, in terms of (i) modeling strategy (point vs. sequence), (ii) type of prior distribution used (fixed Gaussian vs. learned Gaussian Mixture) and (iii) anomaly scoring strategy (based on prior vs. aggregated posterior). Like in the last chapter, we highlight the 14 conclusions drawn from these experiments as **C1-14**. Finally, Section 5.4 applies our DIVAD framework to the Application Server Dataset (ASD), showing that its explicit domain generalization is more broadly applicable beyond our Spark Streaming dataset.

5.1 Anomaly Detection under Domain Shift

This section formally characterizes the problem of anomaly detection under domain shift, as well as the domain generalization framework we adopt to address it.

One of the first adopted definitions of an *anomaly* was proposed by Douglas M. Hawkins in 1980, describing it as "an observation which deviates so much from the other observations as to arouse suspicions that it was *generated by a different mechanism*" [118]. This definition naturally suggests addressing the problem of Section 4.1 from a *generative* perspective, where we assume the labeled windows from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, resulting from the data windowing step described in Section 3.3.3, were all generated from a distribution $p_{\text{data}}(\mathbf{x}, \text{y})$[1], with *normal* windows generated from $p_{\text{data}}(\mathbf{x}|\text{y} = 0)$. In anomaly detection, our general goal then translates to constructing a model $p_{\boldsymbol{\theta}}(\mathbf{x})$ of $p_{\text{data}}(\mathbf{x}|\text{y} = 0)$, parameterized by $\boldsymbol{\theta} \in \Theta$. This yields a natural definition for the anomaly score of a test sample, as its negative log-likelihood with respect to this model:

$$\boxed{g_W(\boldsymbol{x}; \boldsymbol{\theta}) := -\log p_{\boldsymbol{\theta}}(\mathbf{x} = \boldsymbol{x})}$$

Since normal samples from the training and test sets are assumed generated from $p_{\text{data}}(\mathbf{x}|\text{y} = 0) \approx p_{\boldsymbol{\theta}}(\mathbf{x})$, we would indeed expect them to have a higher likelihood under this model than anomalous windows, generated from $p_{\text{data}}(\mathbf{x}|\text{y} = 1) \neq p_{\boldsymbol{\theta}}(\mathbf{x})$.

A specificity of our AIOps use case, however, is that the distribution generating an observed sample can not only be conditioned on its class, but also on the specific *sequence* this sample was extracted from. In particular, each sequence corresponds to a *context* that impacts the distribution of observed data, even for a same entity being recorded. These

[1] To simplify the notation, we use $\boldsymbol{x}$ and y to refer to an input window and its corresponding label in the following chapters, instead of $\boldsymbol{X}$ and $\tilde{y}$ used in Section 3.3.3.

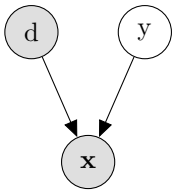


Figure 5.1: Generative model: the observed variable $\mathbf{x}$ depends on its domain d and latent (i.e., unobserved) class y.

contexts can be included in our generative model, by assuming the selection of a sequence i corresponds to the realization d_i of a discrete random variable $\mathrm{d} \sim p_{\text{data}}(\mathrm{d})$ with infinite support[2]. In this setting, the samples of class $c \in \{0, 1\}$ from sequence i:

$$\{\boldsymbol{x}_t^{(i)} \mid y_t^{(i)} = c\}_{t=L}^{T} =: \{(\boldsymbol{x}_c)_t^{(i)}\}_{t=1}^{T_c^{(i)}},$$

can be seen as independently drawn from a sequence-induced, or *domain* distribution:

$$p_i(\mathbf{x}|\mathrm{y} = c) = p_{\text{data}}(\mathbf{x}|\mathrm{y} = c, \mathrm{d} = d_i).$$

This formulation amounts to assuming the distribution of $\mathbf{x}$ is conditioned on the two independent variables d and y, the former determining the domain the sample originates from, and the latter determining whether the sample is normal or anomalous. We illustrate the corresponding generative model of the data in Figure 5.1. Under this model, the data-generating distribution of normal samples can be expressed as the countable mixture of all possible domain distributions:

$$p_{\text{data}}(\mathbf{x}|\mathrm{y} = 0) = \sum_{d=1}^{\infty} p_{\text{data}}(\mathbf{x}|\mathrm{y} = 0, \mathrm{d} = d) p_{\text{data}}(\mathrm{d} = d).$$

Definition 1 (Domain Shift Challenge) *Directly applying traditional generative methods in an unsupervised setting amounts to making $p_{\boldsymbol{\theta}}(\mathbf{x})$ estimate the data-generating distribution of the normal training samples:*

$$p_{\text{train}}(\mathbf{x}|\mathrm{y} = 0) = \frac{1}{N_1} \sum_{i=1}^{N_1} p_{\text{data}}(\mathbf{x}|\mathrm{y} = 0, \mathrm{d} = d_i),$$

with d_i's fixed and all samples equally-likely to come from every sequence i. Given the infinitude of possible domains, this distribution is likely to differ from the data-generating distribution of the normal test samples:

$$p_{\text{test}}(\mathbf{x}|\mathrm{y} = 0) = \frac{1}{N_2} \sum_{i=N_1+1}^{N_1+N_2} p_{\text{data}}(\mathbf{x}|\mathrm{y} = 0, \mathrm{d} = d_i),$$

with $\{d_i\}_{i=1}^{N_1} \neq \{d_i\}_{i=N_1+1}^{N_1+N_2}$. This mismatch induces a domain shift challenge*, characterized by test normal samples $\boldsymbol{x}_0 \sim p_{\text{test}}(\mathbf{x}|\mathrm{y} = 0)$ and test anomalies $\boldsymbol{x}_1 \sim p_{\text{test}}(\mathbf{x}|\mathrm{y} = 1)$ being both unlikely in uncontrollable ways under $p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p_{\text{train}}(\mathbf{x}|\mathrm{y} = 0)$, which hinders anomaly detection performance.*

[2]We use d_i (as opposed to i) to reflect the fact that multiple sequences can correspond to the same context, and thus domain value d (i.e., we can have $d_i = d_j$ for $i \neq j$).

5.1.1 Domain Generalization Framework

A suitable framework to address this domain shift challenge is *domain generalization* [20, 21]. In this framework, the domains sampled for training are referred to as *source domains*, while those sampled at test time are called *target domains*.

Definition 2 (Anomaly Detection with Domain Generalization) *Our problem can be framed as* building an anomaly detection model from the source domains that generalizes to the target domains. *We do so by assuming that the* observed *variable* $\mathbf{x}$ *can be mapped via* f_y *to a* latent *representation* $\mathbf{z}_y$*, whose distribution is (i) discriminative with respect to the class* y *(i.e., normal vs. anomalous) and (ii) independent from the domain* d*. Our goal can be formulated as:*

- *Finding such a mapping* $f_y(\mathbf{x}) = \mathbf{z}_y$*;*
- *Constructing* $p_{\boldsymbol{\theta}}(\mathbf{x})$ *to estimate* $p_{\text{train}}(f_y(\mathbf{x})|\mathrm{y}=0)$ *instead of* $p_{\text{train}}(\mathbf{x}|\mathrm{y}=0)$.

Since $f_y(\mathbf{x}) = \mathbf{z}_y$ is independent from d, we then have:

$$p_{\text{train}}(\mathbf{z}_y|\mathrm{y}=0) = \frac{1}{N_1}\sum_{i=1}^{N_1} p(\mathbf{z}_y|\mathrm{y}=0, \mathrm{d}=d_i)$$
$$p_{\text{train}}(\mathbf{z}_y|\mathrm{y}=0) = p(\mathbf{z}_y|\mathrm{y}=0) = p_{\text{test}}(\mathbf{z}_y|\mathrm{y}=0),$$

which means that, under $p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p_{\text{train}}(\mathbf{z}_y|\mathrm{y}=0) = p_{\text{test}}(\mathbf{z}_y|\mathrm{y}=0)$, the **normal test samples $\boldsymbol{x}_0$ should be more likely than the test anomalies $\boldsymbol{x}_1$**, hence addressing the domain shift challenge.

5.2 Domain-Invariant VAE for Anomaly Detection

This section introduces our new approach to anomaly detection under domain shift. At a high level, its central assumption is that *anomalies should have a sensible impact on the properties of the input samples that are invariant with respect to the domain*. In an AIOps scenario, this means that, although some aspects of a running process may vary from domain to domain, others typically remain constant and characterize its "normal" behavior. These domain-invariant, normal-specific characteristics can be seen as reflecting whether the process *functions properly*, while domain-specific characteristics simply manifest *different modes of normal operation*. Such "invariant" behaviors can for instance relate to the Key Performance Indicators (KPIs) [9] of a process, which semantically reflect its well-functioning or overall health, and should therefore behave similarly across different contexts.

To construct the mapping f_y defined in Section 5.1.1, we propose Domain-Invariant VAE for Anomaly Detection (DIVAD). This method relies on *feature disentanglement* [20, 21], assuming that the observed variable $\mathbf{x}$ is caused by two *independent* latent factors $\mathbf{z}_d$ and $\mathbf{z}_y$, where $\mathbf{z}_d$ is conditioned on the observed domain d, while $\mathbf{z}_y$ is assumed independent from it and can be used to detect anomalies in test samples. The corresponding generative model is shown in Figure 5.2. Further, we assume this model is parameterized by *model parameters* $\boldsymbol{\theta} \in \Theta$, and that the "complex", intractable, marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathrm{d})$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathrm{d}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_d, \mathbf{z}_y|\mathrm{d}) d\mathbf{z}_d d\mathbf{z}_y$$
$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathrm{d}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_y, \mathbf{z}_d) p_{\boldsymbol{\theta}}(\mathbf{z}_d|\mathrm{d}) p(\mathbf{z}_y) d\mathbf{z}_d d\mathbf{z}_y,$$

can be constructed from "simpler", tractable prior and likelihood distributions, whose parameters can be complex functions of their inputs.

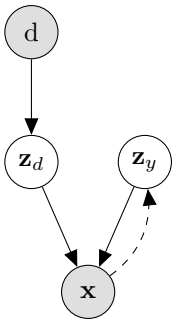


Figure 5.2: Generative model: $\mathbf{x}$ is caused by independent domain-specific $\mathbf{z}_d$ and domain-independent $\mathbf{z}_y$. Constructing f_y then amounts to *inferring* $\mathbf{z}_y$ from $\mathbf{x}$ (dashed arrow).

5.2.1 Model Training

To uncover the latent factors, and ultimately $p_{\boldsymbol{\theta}}(\mathbf{z}_y|\mathbf{x})$, our goal is to adjust the model parameters to best fit the training data. To do so, we leverage amortized variational inference using the reparameterization trick (i.e., consider a variational autoencoder (**VAE**) **framework** [119, 120], as described in Appendix B). In this framework, we consider additional *variational parameters* $\boldsymbol{\phi}_d, \boldsymbol{\phi}_y \in \Phi$, and optimize the evidence lower bound (**ELBO**) with KL divergence terms weighted by a factor β [121, 71]:

$$\begin{aligned}\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}, d; \boldsymbol{\theta}_{yd}, \boldsymbol{\theta}_d, \boldsymbol{\phi}_d, \boldsymbol{\phi}_y) = \mathbb{E}_{q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x})q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}_{yd}}(\boldsymbol{x}|\mathbf{z}_d, \mathbf{z}_y)] \\ - \beta D_{\text{KL}}(q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x})\|p(\mathbf{z}_y)) - \beta D_{\text{KL}}(q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x})\|p_{\boldsymbol{\theta}_d}(\mathbf{z}_d|d)),\end{aligned} \tag{5.1}$$

with:

$$\begin{aligned}p_{\boldsymbol{\theta}_{yd}}(\mathbf{x}|\mathbf{z}_d, \mathbf{z}_y) &= \mathcal{N}(\text{NN}_{\boldsymbol{\theta}_{yd}}(\mathbf{z}_d, \mathbf{z}_y), \text{NN}_{\boldsymbol{\theta}_{yd}}(\mathbf{z}_d, \mathbf{z}_y)) \\ p_{\boldsymbol{\theta}_d}(\mathbf{z}_d|\text{d}) &= \mathcal{N}(\text{NN}_{\boldsymbol{\theta}_d}(\text{d}), \text{NN}_{\boldsymbol{\theta}_d}(\text{d})) \\ q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x}) &= \mathcal{N}(\text{NN}_{\boldsymbol{\phi}_y}(\mathbf{x}), \text{NN}_{\boldsymbol{\phi}_y}(\mathbf{x})) \\ q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\mathbf{x}) &= \mathcal{N}(\text{NN}_{\boldsymbol{\phi}_d}(\mathbf{x}), \text{NN}_{\boldsymbol{\phi}_d}(\mathbf{x}))\end{aligned}$$

where $\mathcal{N}(\text{NN}_{\boldsymbol{\theta}}(\cdot), \text{NN}_{\boldsymbol{\theta}}(\cdot))$ denotes a multivariate Gaussian distribution whose mean and variance are outputted by a neural network with parameters $\boldsymbol{\theta}$, and $p(\mathbf{z}_y)$ is the $\mathbf{z}_y$ *prior*, used for anomaly scoring and further detailed in Section 5.2.2. The conditional prior $p_{\boldsymbol{\theta}_d}(\mathbf{z}_d|\text{d})$ has the effect of making $\mathbf{z}_d$ more dependent on d, by ensuring that signals from d are incorporated into $\mathbf{z}_d$ (and thus facilitating the classification of d given $\mathbf{z}_d$).

To further enforce this ease of classification, we add to maximum likelihood the following **domain discrimination** objective:

$$\mathcal{L}_d(\boldsymbol{x}, d; \boldsymbol{\phi}_d, \boldsymbol{\omega}_d) = \mathbb{E}_{q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x})} \log q_{\boldsymbol{\omega}_d}(d|\mathbf{z}_d),$$

with $\boldsymbol{\omega}_d \in \Omega$ the *domain discriminator parameters*. In practice, this objective amounts to training a domain discrimination head by minimizing the cross-entropy loss based on the source domain labels. We finally perform gradient ascent on the overall maximization objective:

$$\mathcal{L}(\boldsymbol{x}, d; \boldsymbol{\theta}_{yd}, \boldsymbol{\theta}_d, \boldsymbol{\phi}_d, \boldsymbol{\phi}_y, \boldsymbol{\omega}_d) = \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}, d; \boldsymbol{\theta}_{yd}, \boldsymbol{\theta}_d, \boldsymbol{\phi}_d, \boldsymbol{\phi}_y) + \alpha_d \mathcal{L}_d(\boldsymbol{x}, d; \boldsymbol{\phi}_d, \boldsymbol{\omega}_d),$$

with $\alpha_d \in \mathbb{R}$ a tradeoff hyperparameter balancing the maximum likelihood estimation of the generative model and the domain discrimination. We do not share the parameters of our encoder networks $\text{NN}_{\boldsymbol{\phi}_y}$ and $\text{NN}_{\boldsymbol{\phi}_d}$, but instead consider a multi-encoder architecture.

DIVAD is similar in spirit to *Domain-Invariant Variational Autoencoders* (DIVA) [71], proposed for image classification. Our problem setting of unsupervised anomaly detection however leads to major differences from classification-based DIVA. First, by not relying on training class labels, DIVAD fuses DIVA's class-conditioned and residual latent factors $\mathbf{z}_y$ and $\mathbf{z}_x$ into a single, unconditioned domain-invariant factor $\mathbf{z}_y$, considering a conditioning and auxiliary classification objective only for the domain-specific factor $\mathbf{z}_d$. Second, rather than relying on an explicit classifier on top of the class-specific factor $\mathbf{z}_y$, DIVAD derives its anomaly scores from these factor's training distribution, modeled with the flexibility described in Sections 5.2.2 and 5.2.3.

5.2.2 Anomaly Scoring based on Prior

After training, we should have the variational posterior $q_{\phi_y}(\mathbf{z}_y|\mathbf{x})$ approximate the model posterior $p_{\boldsymbol{\theta}}(\mathbf{z}_y|\mathbf{x})$ (dashed arrow in Figure 5.2). We can therefore use this variational posterior to construct our mapping f_y:

$$\mathbf{z}_y = f_y(\mathbf{x}) \sim q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x}) \approx p_{\boldsymbol{\theta}}(\mathbf{z}_y|\mathbf{x}).$$

The **anomaly score** $g_W(\boldsymbol{x})$ of a data sample $\boldsymbol{x}$ can then simply be defined as the negative log-likelihood of $f_y(\boldsymbol{x})$ with respect to the prior $p(\mathbf{z}_y)$:

$$\boxed{g_W(\boldsymbol{x}) := -\log p(\mathbf{z}_y = f_y(\boldsymbol{x}))} \tag{5.2}$$

Indeed, maximizing Equation 5.1 on average on the training set amounts to maximizing the regularization term of the ELBO with respect to $\mathbf{z}_y$:

$$\Omega_{\boldsymbol{\phi}_y} := -\mathbb{E}_{p_{\text{train}}(\mathbf{x})} D_{\text{KL}}(q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x}) \| p(\mathbf{z}_y)),$$

which reads (from [122]):

$$\begin{aligned}
\Omega_{\boldsymbol{\phi}_y} &= -\mathbb{E}_{\hat{p}_{\text{train}}(\mathbf{x})} \mathbb{E}_{q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x})} q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x}) \log \frac{q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x})}{p(\mathbf{z}_y)} \\
&= \int\int \hat{p}_{\text{train}}(\mathbf{x}) q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x}) (\log p(\mathbf{z}_y) - \log q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x})) d\mathbf{x} d\mathbf{z}_y \\
&= \int\int \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \delta(\mathbf{x} - \boldsymbol{x}_i) q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x}) (\log q_{\boldsymbol{\phi}_y}(\log p(\mathbf{z}_y) - \mathbf{z}_y|\mathbf{x})) d\mathbf{x} d\mathbf{z}_y \\
\Omega_{\boldsymbol{\phi}_y} &= \int \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i) \log p(\mathbf{z}_y) d\mathbf{z}_y - \int \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i) \log q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i) d\mathbf{z}_y,
\end{aligned}$$

with N_{train} the total number of training samples, and $\hat{p}_{\text{train}}$ the empirical training distribution. By considering:

$$q_{\boldsymbol{\phi}_y}(\mathbf{z}_y) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i),$$

the **marginal**, or **aggregated posterior** [123, 124] (here the empirical distribution of encoded, presumably domain-invariant, samples), we therefore have:

$$\Omega_{\boldsymbol{\phi}_y} = \int q_{\boldsymbol{\phi}_y}(\mathbf{z}_y) \log p(\mathbf{z}_y) d\mathbf{z}_y - \int \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i) \log q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i) d\mathbf{z}_y$$
$$\Omega_{\boldsymbol{\phi}_y} = -H(q_{\boldsymbol{\phi}_y}(\mathbf{z}_y), p(\mathbf{z}_y)) + H(q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x})),$$

with $H(q_{\boldsymbol{\phi}_y}(\mathbf{z}_y), p(\mathbf{z}_y))$ the cross-entropy between the aggregated posterior and the prior, and $H(q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x}))$ the conditional entropy of $q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x})$ with the empirical distribution $\hat{p}_{\text{train}}(\mathbf{x})$ [122].

As we can see, **the maximization process of the ELBO has the effect of trying to make the aggregated posterior $q_{\boldsymbol{\phi}_y}(\mathbf{z}_y)$ match the prior $p(\mathbf{z}_y)$**, which *a priori* motivates the choice above of using the prior to derive anomaly scores.

Fixed Standard Gaussian Prior

We first consider the default choice of prior for $\mathbf{z}_y$ in a VAE framework, as a fixed standard Gaussian distribution:

$$p(\mathbf{z}_y) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}),$$

and refer to the method that uses this $\mathbf{z}_y$ prior and scores with Equation 5.2 as **DIVAD-G**. A limitation of DIVAD-G is that, although the aggregated posterior and prior *should* be brought closer when maximizing the ELBO, they usually do not end up matching in practice at the end of training [124, 125]. This phenomenon is sometimes described as "*holes in the aggregated posterior*", referring to the regions of the latent space that have high density under the prior but very low density under the aggregated posterior [126].

Learned Gaussian Mixture Prior

A method that has been shown to (at least partly) address the problem of aggregated posterior holes is to replace the fixed $\mathbf{z}_y$ prior with a *learnable prior* $p_{\boldsymbol{\lambda}}(\mathbf{z}_y)$ [126, 122], and hence have the maximization process update both the aggregated posterior and the prior. If sufficiently expressive, the prior can serve as a good approximation $\hat{q}_{\boldsymbol{\phi}_y}(\mathbf{z}_y)$ of the aggregated posterior at the end of training, which makes it safer to use for anomaly scoring:

$$\boxed{g_W(\boldsymbol{x}) := -\log p_{\boldsymbol{\lambda}}(\mathbf{z}_y = f_y(\boldsymbol{x}))} \tag{5.3}$$

In a way, considering a learnable prior amounts to explicitly performing a joint density estimation of the marginal likelihood and aggregated posterior. With sufficiently expressive priors, this joint estimation also has the effect of *putting less constraints on the aggregated posterior*, letting it capture normal clusters with more variance and arbitrary shapes. This is particularly useful in our anomaly detection context, where the "normal" class is an umbrella term for a variety of different normal behaviors (even for a same monitored entity). In practice, any density estimator $p_{\boldsymbol{\lambda}}(\mathbf{z}_y)$ can be used to model the aggregated posterior. In this work, we consider a Gaussian Mixture (GM) distribution with K components:

$$p_{\boldsymbol{\lambda}}(\mathbf{z}_y) = \sum_{k=1}^{K} w_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2),$$

with $\boldsymbol{\lambda} = \{w_k, \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2\}_{k=1}^K$ randomly initialized and trained along with the other parameters. We refer to the method that uses this $\mathbf{z}_y$ prior and scores with Equation 5.3 as **DIVAD-GM**.

5.2.3 Anomaly Scoring based on Aggregated Posterior Estimate

An alternative (or complementary) solution to the problem of aggregated posterior holes is to perform the density estimation of the aggregated posterior $\hat{q}_{\boldsymbol{\phi}_y}(\mathbf{z}_y)$ separately, and then define the anomaly score with respect to this estimate instead of the prior:

$$\boxed{g_W(\boldsymbol{x}) := -\log \hat{q}_{\phi_y}(\mathbf{z}_y = f_y(\boldsymbol{x}))} \tag{5.4}$$

In the following experiments, we consider this alternative in addition to the prior-based scoring for both DIVAD-G and DIVAD-GM. For DIVAD-G, the aggregated posterior is performed by fitting a multivariate Gaussian distribution to the training samples in latent space. For DIVAD-GM, it is performed by fitting to them a Gaussian Mixture model with the same number of components K as the prior.

5.2.4 Putting It All Together

We illustrate the multi-encoder architecture of our DIVAD method in Figure 5.3, shown here for the learned Gaussian Mixture prior detailed in Section 5.2.2. From this figure, we can see that encoder networks $\text{NN}_{\boldsymbol{\phi}_d}$ and $\text{NN}_{\boldsymbol{\phi}_y}$ take the same sample $\boldsymbol{x}$ as input to output the mean and variance parameters of multivariate Gaussians $q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\mathbf{x})$ and $q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\mathbf{x})$, respectively. These parameters are first used to compute the KL divergence terms of Equation 5.1, with the parameters of the conditional prior $p_{\boldsymbol{\theta}_d}(\mathbf{z}_d|\text{d})$ outputted by a network $\text{NN}_{\boldsymbol{\theta}_d}$ from the domain d of $\boldsymbol{x}$, and the parameters of $p_{\boldsymbol{\lambda}}(\mathbf{z}_y)$ learned as described in Section 5.2.2. They are then used to sample the corresponding domain and class encodings of $\boldsymbol{x}$: $\boldsymbol{z}_d$ and $\boldsymbol{z}_y$. These encodings, of same dimension M', are further concatenated to form the input of the decoder network $\text{NN}_{\boldsymbol{\theta}_{yd}}$, outputting the parameters of the multivariate Gaussian $p_{\boldsymbol{\theta}_{yd}}(\mathbf{x}|\mathbf{z}_d, \mathbf{z}_y)$, from which the likelihood (or *reconstruction*) term of Equation 5.1 is computed. The bottom right of the figure finally shows the domain discrimination head, $\text{NN}_{\boldsymbol{\omega}_d}$, which takes the domain encoding $\boldsymbol{z}_d$ of $\boldsymbol{x}$ as input, and outputs the parameters of the Categorical $q_{\boldsymbol{\omega}_d}(\text{d}|\mathbf{z}_d)$, used to compute the domain discrimination objective $\mathcal{L}_d$.

5.3 Experiments

This section applies the DIVAD-G and DIVAD-GM methods introduced in Section 5.2 against the Exathlon benchmark and dataset. We use the experimental setup described in Section 4.2, with the training windows balanced by *domain* instead of (application, Spark settings, input rate) triplet. We start by defining the concept of domain in our experimental setup, outlining our compared DIVAD variants and their grid of hyperparameter values. We then present and analyze the results obtained, deriving 14 conclusions labeled **C1-14**, mostly about the (i) effectiveness of the domain generalization performed and the improvement it induced on the detection performance, the (ii) benefits and limitations of point and sequence modeling methods and their ability to detect different event types, and the (iii) effect the anomaly scoring strategy had on the performance according to the DIVAD variant.

5.3.1 Compared Methods and Hyperparameters

In our experimental setup, we do not require the methods to generalize to new Spark Streaming *applications*, but rather to the *context* the eight applications seen in training were recorded in. For this reason, we simply define the *domain* of a trace as its context, which, from Section 4.3, corresponds to the Spark settings and input rate used for its

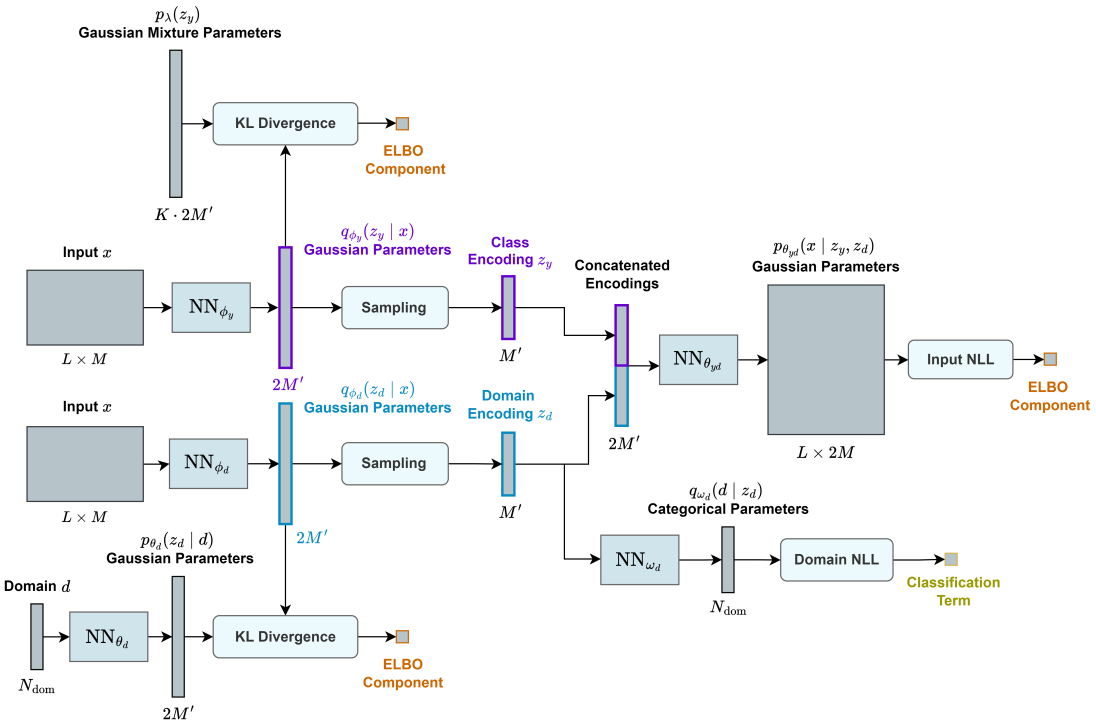


Figure 5.3: Multi-encoder architecture of our DIVAD-GM models, with N_{dom} the number of training domains (DIVAD-G models use a similar architecture, with the learned Gaussian Mixture parameters replaced with fixed Gaussian parameters).

application run. Using this definition, we obtain **22 source domains** and **11 target domains** to consider for our DIVAD variants.

We adopt the same model training and selection strategy as described in Section 4.4.1 for both DIVAD-G and DIVAD-GM. For both variants, we use the same strategy as Dense VAE and Rec VAE for deriving encoding and output standard deviations. We first consider *point modeling* DIVAD methods ($L = 1$), using fully-connected neural network architectures and referred to as **Dense DIVAD-G** and **Dense DIVAD-GM**, respectively. We report the performance of Dense DIVAD-G using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- A single hidden layer of 200 units for the encoders $\text{NN}_{\boldsymbol{\phi}_y}$ and $\text{NN}_{\boldsymbol{\phi}_d}$ as well as for the decoder $\text{NN}_{\boldsymbol{\theta}_{yd}}$.
- A single fully-connected hidden layer of 64 units for the conditional domain prior network $\text{NN}_{\boldsymbol{\theta}_d}$.
- A ReLU activation function followed by a single fully-connected layer of $N_{\text{dom}} = 22$ units (i.e., one per source domain) for the classification head $\text{NN}_{\boldsymbol{\omega}_d}$. The domain encoding $\boldsymbol{z}_d$ is indeed sampled unbounded from $q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x})$, so we pass it through a ReLU activation before applying the output layer. The classification head was intentionally kept simple, so as to be able to *easily* classify the domain from $\boldsymbol{z}_d$.
- The ReLU activation function for all the layers except the encoding and output layers.
- An encoding dimension in $\{16, 64\}$.
- A fixed KL divergence weight $\beta \in \{1, 5\}$. We also tried using the linear scheduling strategy of DIVA [71], but this did not lead to a better performance.

- A domain classification weight $\alpha_d = 100,000$. We set this weight based on the scale we observed for our losses $\mathcal{L}_{\text{ELBO}}$ and $\mathcal{L}_d$ during training in some initial experiments. We also tried different weight values, as well as the CoV-Weighting strategy proposed by Groenendijk et al. [127] to automatically balance these two losses, but this did not lead to a better performance.
- A mini-batch size $B = 128$.
- An anomaly scoring based on the class encoding aggregated posterior estimate $\hat{q}_{\boldsymbol{\phi}_y}(\mathbf{z}_y)$, fitting a multivariate Gaussian distribution to the training class encodings.

We report the performance of Dense DIVAD-GM using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- The same architectures as Dense DIVAD-G for the networks $\text{NN}_{\boldsymbol{\phi}_y}$, $\text{NN}_{\boldsymbol{\phi}_d}$, $\text{NN}_{\boldsymbol{\theta}_{yd}}$, $\text{NN}_{\boldsymbol{\theta}_d}$ and $\text{NN}_{\boldsymbol{\omega}_d}$.
- An encoding dimension in $\{16, 32\}$, with $K = 8$ Gaussian Mixture components when using an encoding dimension of 16, and $K = 4$ components when using an encoding dimension of 32.
- A fixed KL divergence weight $\beta \in \{1, 5\}$.
- A domain classification weight $\alpha_d = 100,000$.
- A mini-batch size $B = 128$.
- An anomaly scoring based on (i) the learned class encoding prior $p_{\boldsymbol{\lambda}}(\mathbf{z}_y)$, and (ii) the class encoding aggregated posterior estimate $\hat{q}_{\boldsymbol{\phi}_y}(\mathbf{z}_y)$, fitting a Gaussian Mixture distribution with K components to the training class encodings.

We also consider *sequence modeling* DIVAD methods ($L = 20$ here), with some fully-connected neural network architectures replaced by recurrent ones based on the design of Figure 4.8, referred to as **Rec DIVAD-G** and **Rec DIVAD-GM**, respectively. We report the performance of Rec DIVAD-G using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- Each encoder $\text{NN}_{\boldsymbol{\phi}_y}$ and $\text{NN}_{\boldsymbol{\phi}_d}$ with a 1D convolutional layer using 64 filters of size 5, a stride length of 1 and the ReLU activation function, followed by a GRU layer of 64 units using the tanh activation function, and a fully-connected layer to output the encoding parameters.
- The decoder $\text{NN}_{\boldsymbol{\theta}_{yd}}$ defined symmetrically to one encoder as per the design of Figure 4.8.
- The same architectures as Dense DIVAD-G for the conditional domain prior network $\text{NN}_{\boldsymbol{\theta}_d}$ and classification head $\text{NN}_{\boldsymbol{\omega}_d}$.
- An encoding dimension of 32.
- A fixed KL divergence weight $\beta \in \{1, 5\}$.
- A domain classification weight $\alpha_d = 100,000$.
- A mini-batch size $B = 128$.
- The same anomaly scoring as Dense DIVAD-G.

We report the performance of Rec DIVAD-GM using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- The same architectures as Rec DIVAD-G for the networks $\text{NN}_{\boldsymbol{\phi}_y}$, $\text{NN}_{\boldsymbol{\phi}_d}$, $\text{NN}_{\boldsymbol{\theta}_{yd}}$, $\text{NN}_{\boldsymbol{\theta}_d}$ and $\text{NN}_{\boldsymbol{\omega}_d}$.
- An encoding dimension of 32, with $K = 8$ Gaussian Mixture components.
- A fixed KL divergence weight $\beta \in \{1, 5\}$.

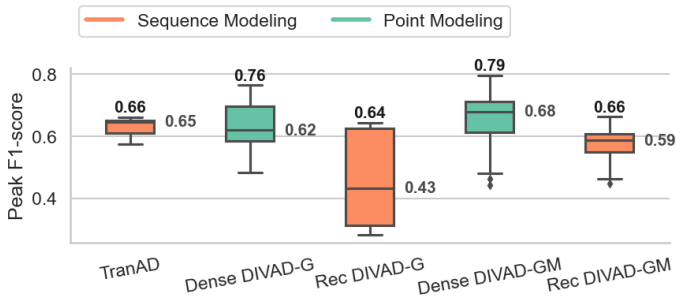


Figure 5.4: Box plots of peak F1-scores achieved by TranAD and each DIVAD variant, colored by modeling strategy (point vs. sequence).

Method	Tr-T1	Tr-T2	Tr-T3	Tr-T4	Tr-T5	Tr-T6
Dense DIVAD-GM	**0.80**	0.97	**0.87**	0.73	**0.96**	0.57
Dense DIVAD-G	0.78	**0.98**	0.79	**0.75**	0.93	0.53
Point Average	0.79	0.98	**0.83**	0.74	**0.95**	0.55
Rec DIVAD-GM	0.78	0.97	0.76	0.72	0.75	**0.68**
Rec DIVAD-G	**0.53**	0.97	0.73	0.69	0.72	0.58
Sequence Average	0.66	0.97	**0.75**	0.71	**0.74**	0.63
Average	0.72	**0.97**	**0.79**	0.72	**0.84**	**0.59**

Table 5.1: Peak F1-score achieved by the best-performing DIVAD variants for each event type within a trace (averaged across test traces), with the top F1-score for each event type shown in **bold**.

- A domain classification weight $\alpha_d = 100,000$.
- A mini-batch size $B = 128$.
- The same anomaly scoring as Dense DIVAD-GM.

5.3.2 Results and Analyses

We show in Figure 5.4 the box plots of the peak F1-scores achieved by each DIVAD variant across its hyperparameter values, with boxes colored based on the modeling strategy (point vs. sequence). We also include the performance of TranAD for reference, as our best-performing unsupervised baseline from Chapter 4. Table 5.1 follows the same format as Table 4.1 described in Section 4.5.1. It considers our DIVAD variants with their best-performing hyperparameters (i.e., that yielded their maximum peak F1-score in Figure 5.4), and reports the average peak F1-score they obtained within each test trace for each type of event.

Improvements over Unsupervised Baselines

The main observation we can make from Figure 5.4 is that Dense DIVAD-GM and Dense DIVAD-G significantly outperformed our best unsupervised baseline in maximum performance (**C1**), with 20% and 15% improvements in maximum peak F1-scores (0.79 and 0.76 over 0.66), respectively. As expected, resorting to a learned Gaussian Mixture prior (DIVAD-GM) instead of a fixed Gaussian prior (DIVAD-G) was also beneficial, improving both the maximum and median peak F1-scores for the point and sequence modeling variants (**C2**).

Although Dense DIVAD-GM could outperform TranAD in the median (with a less-

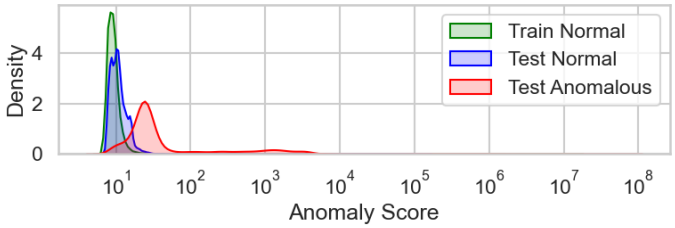


Figure 5.5: Kernel Density Estimate (KDE) plots of the anomaly scores assigned by Dense DIVAD-GM to training normal, test normal and test anomalous records.

significant 3% improvement), the general trend was to get a lower median F1-score for the DIVAD methods than for TranAD. Overall, the large differences observed between the maximum and median F1-scores of the DIVAD variants reveals their particular sensitivity to the hyperparameter values used, and the importance of properly tuning them to benefit from a performance gain in practice (**C3**).

The higher performance achieved by Dense DIVAD-GM can directly be attributed to its accurate domain generalization (**C4**). To illustrate this, Figure 5.5 shows the Kernel Density Estimate (KDE) plots of the anomaly scores the best-performing Dense DIVAD-GM assigned to the training normal, test normal and test anomalous records. From this figure, we can see that the explicit modeling of the training data distribution carried out by Dense DIVAD-GM led to a similar benefit as Dense VAE (see Figure 4.10d), with a low variance in the anomaly scores assigned to the training normal records (i.e., narrow **green** KDE). Contrary to Dense VAE, however, Dense DIVAD-GM relied on this precise density estimation in a *domain-invariant* space (where distribution shifts were drastically reduced), which made it generalize to *test* normal records as well (i.e., better *aligned* and similarly narrow **green** and **blue** KDEs). As such, Dense DIVAD-GM could generally view the test anomalies as "more abnormal" than the test normal records (i.e., good separation of the **red** and **blue** KDEs), which led to the better anomaly detection performance.

The accurate domain generalization of the best-performing Dense DIVAD-GM is also illustrated in Figure 5.6, showing the ridgeline plot of its anomaly scores for the records of each type within the 23 test traces, as well as its peak F1-score threshold (akin to Figures 4.13 and 4.14 for the TranAD and Rec DSVDD methods, respectively). From this figure, we can see that the anomaly scores assigned by Dense DIVAD-GM to the test *normal* records were (i) mostly lower than the anomaly scores it assigned to anomalies, but also (ii) far more similar across *different traces* than for TranAD (see Figure 4.13), allowing Dense DIVAD-GM to achieve a superior detection performance using a *single*, *global threshold*. Moreover, the *explicit* domain generalization of Dense DIVAD-GM was more effective than the implicit one produced by Rec DSVDD (see Figure 4.14), with for instance most of the T1 records of traces `5_1_100000_63`, `5_1_100000_64` and `5_1_500000_62`, which were missed by Rec DSVDD, now being detected above the optimal threshold.

To further illustrate Dense DIVAD-GM's accurate comprehension of test domains, Figure 5.7 shows t-SNE scatter plots of the domain-specific and domain-invariant encodings it produced for test normal records (sampled from $q_{\phi_d}(\mathbf{z}_d|\mathbf{x})$ and $q_{\phi_y}(\mathbf{z}_y|\mathbf{x})$, respectively), undersampled to 10,000 data records, balanced and colored by domain. Domain labels follow the same format as described in Section 4.3.2, indicating the processing period, number of Spark executors, maximum executors memory and data sender input rate, respectively. From this figure, we can see that the mapping learned by Dense DIVAD-GM from the *input* to its *domain-specific* space produced the distinct domain clusters expected, while the mapping it learned from the *input* to its *domain-invariant* space produced more scattered encodings.

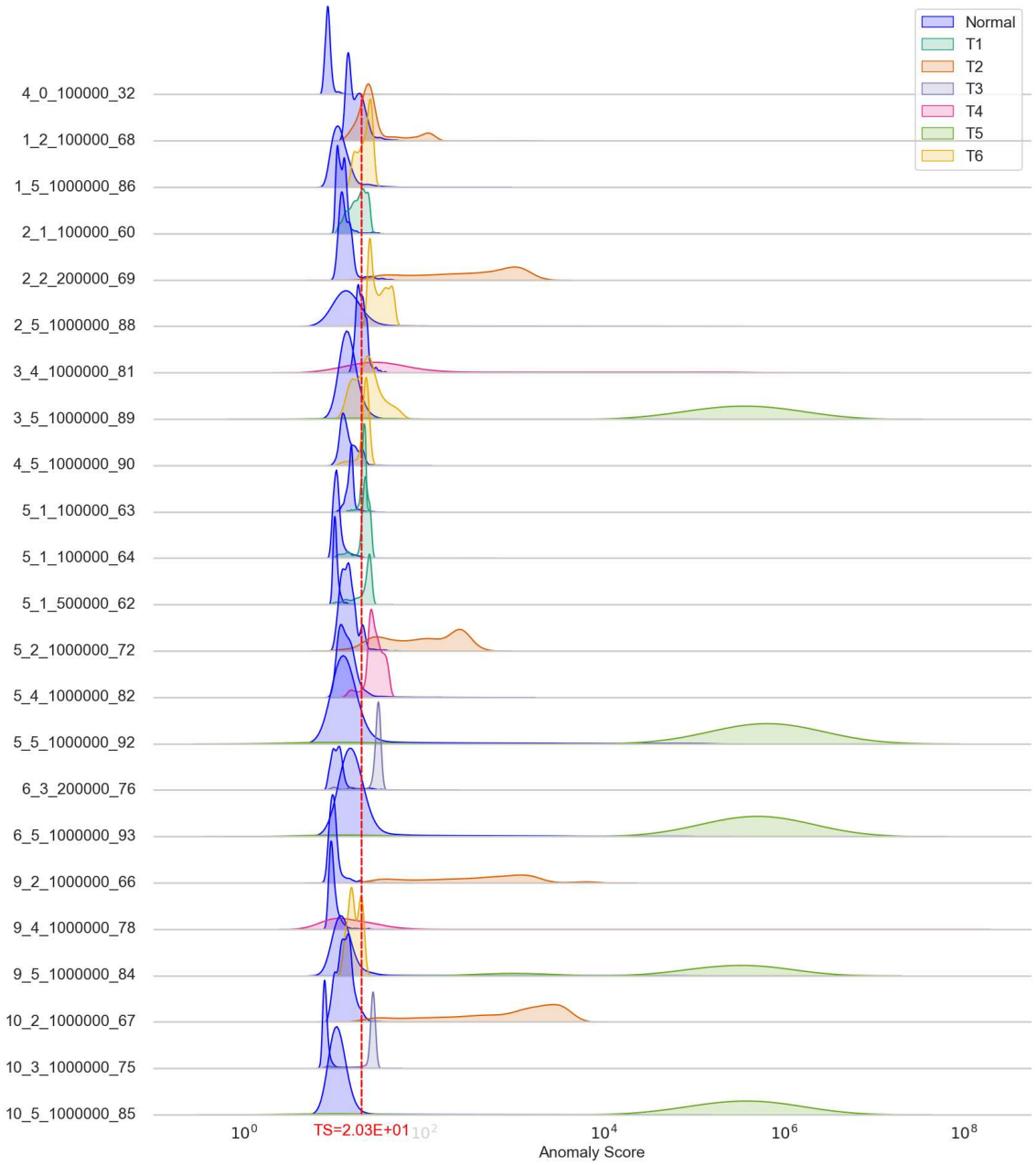


Figure 5.6: Ridgeline plot of Dense DIVAD-GM's anomaly scores for the records of each type (i.e., in "normal" and T1 to T6 ranges) in each test trace, with its peak F1-score threshold highlighted in red.

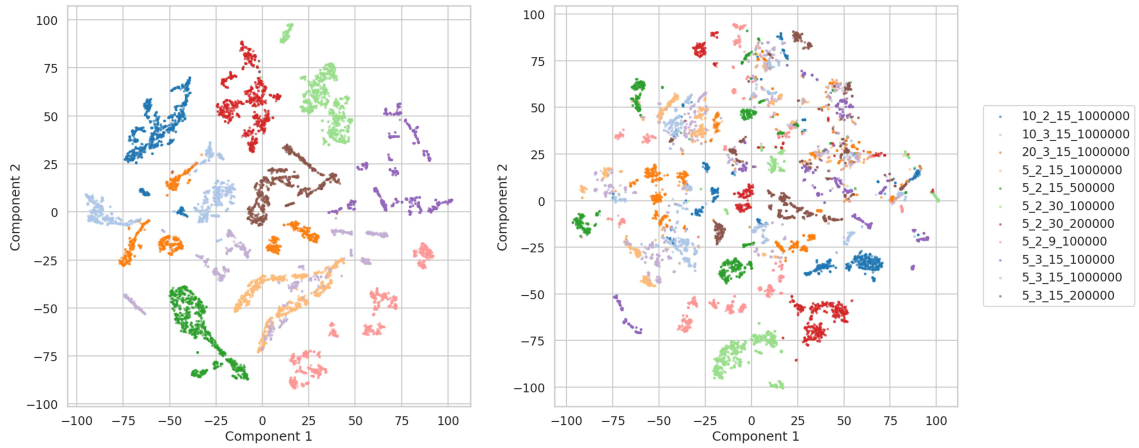


Figure 5.7: t-SNE scatter plots of Dense DIVAD-GM's domain-specific (left) and domain-invariant (right) encodings of test normal records, undersampled to 10,000 data records, balanced and colored by domain.

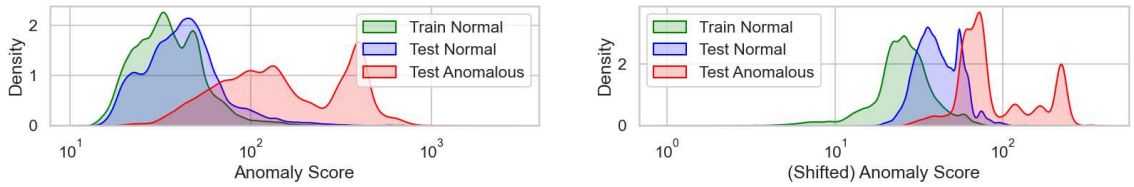
Point vs. Sequence Modeling

Another observation we can make from Figure 5.4 is that using point modeling over sequence modeling DIVAD variants appeared both *sufficient* (**C5**) and *necessary* (**C6**) to outperform TranAD for our dataset and experimental setup.

Point modeling variants being *sufficient* to outperform our unsupervised baselines is consistent with our description of Section 4.3, stating that nearly all the event types were mainly reflected in the features as *contextual* anomalies, which could typically be turned into (easier-to-detect) *point* anomalies when viewed in the domain-invariant spaces of the DIVAD methods. Referring back to the central assumption of our DIVAD framework (discussed at the start of Section 5.2), considering the values of *feature combinations* at *single time steps* at a time was here sufficient for the point modeling methods to learn domain-invariant patterns that were also discriminative for most anomaly types.

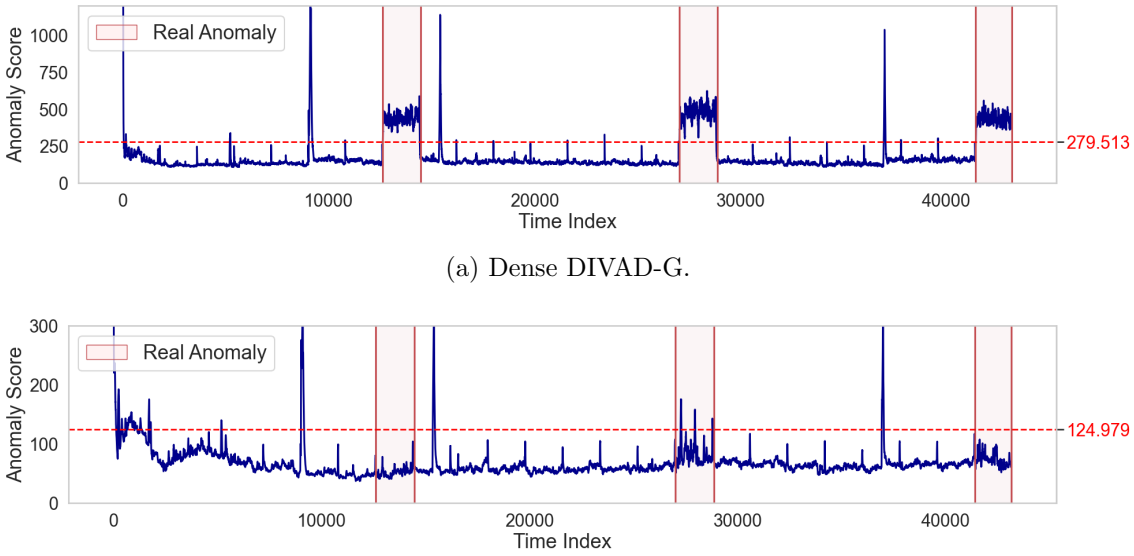
The lower performance observed for sequence modeling DIVAD variants (i.e., point modeling methods being "necessary" in this case) was somewhat less expected *a priori*, but can be explained by two main factors.

The first factor comes from the *unsupervised* nature of the DIVAD methods. Without information on the relevant anomalies to detect, these methods do not have the *incentive* to learn domain-invariant patterns that will *also* accurately distinguish anomalies. For point modeling variants, the feature combinations that tended to be domain-invariant were also useful to detect the anomalies of our dataset and experimental setup. For Rec DIVAD-G, however, the *sequential* patterns learned to be shared across domains also tended to be shared between normal data and specific anomaly types. This is for instance illustrated in Figure 5.8a, showing KDE plots of the anomaly scores assigned by Rec DIVAD-G to training normal, test normal and test anomalous records. From this figure, we can see that Rec DIVAD-G could accurately perform its domain generalization task, with training and test normal records getting assigned similar anomaly scores (i.e., aligned **green** and **blue** KDEs). However, this accurate domain generalization did not result in a better anomaly detection performance, primarily due to Rec DIVAD-G's inability to distinguish some anomalous records from normal data in its domain-invariant space (i.e., high overlap between the **blue** and **red** KDEs). Figure 5.9, showing time plots of the anomaly scores assigned by Dense DIVAD-G and Rec DIVAD-G in trace `5_1_100000_63` (Bursty Input), further illustrates this aspect for T1 events specifically. From these figures, we can see



(a) Rec DIVAD-G. (b) Rec DIVAD-GM.

Figure 5.8: Kernel Density Estimate (KDE) plots of the anomaly scores assigned by Rec DIVAD-G and Rec DIVAD-GM to training normal, test normal and test anomalous records.



(a) Dense DIVAD-G.

(b) Rec DIVAD-G.

Figure 5.9: Time plots of the anomaly scores of Dense DIVAD-G and Rec DIVAD-G for the records in trace `5_1_100000_63` (Bursty Input), highlighting their peak F1-score thresholds and the ground-truth anomaly ranges.

that, while Dense DIVAD-G could accurately view T1 records as "more abnormal" than most normal records in the trace, the encoding performed by Rec DIVAD-G tended to "erase" most of the anomalous signals from these T1 events.

The second factor explaining the lower performance of sequence modeling DIVAD variants is the heightened challenge of learning domain-invariant patterns from the data that are *sequential* in nature. While leveraging such sequential information can theoretically be useful, identifying domain-invariant *shapes* within and across $M = 237$ time series constitutes a harder task than relying on simple feature combinations at given time steps for our dataset and experimental setup, typically requiring more hyperparameter tuning. This is illustrated in Figure 5.8b, showing KDE plots of Rec DIVAD-GM's anomaly scores for training normal, test normal and test anomalous records. From this figure, we can see that the domain generalization performed by Rec DIVAD-GM was less effective than for Dense DIVAD-GM and Rec DIVAD-G (see Figures 5.5 and 5.8b, respectively), with its anomaly scores *drifting* from the training to the test normal records (**green** vs. **blue** KDEs). This suboptimal domain generalization led to a higher overlap between the anomaly scores of the test normal and anomalous records (**blue** vs. **red** KDEs), which resulted in the low anomaly detection performance.

Difficulty of Event Types

Now considering the `Average` row of Table 5.1, we can see that **T2** (Bursty Input Until Crash), **T3** (Stalled Input) and **T5** (Driver Failure) were the **easiest types** of events to detect within test traces across the DIVAD variants (**C7**). The worse detection of **T1** (Bursty Input) events compared to the unsupervised baselines of Chapter 4 (0.72 vs. 0.80 average peak F1-score in Table 4.1) was however only due to Rec DIVAD-G, whose domain generalization tended to remove some of the T1 anomaly signals, as we just discussed. Excluding Rec DIVAD-G, the average T1 performance across DIVAD variants indeed becomes nearly the same as our unsupervised baselines (0.79 and 0.80 peak F1-scores, respectively). The better detection of **T3** (Stalled Input) events by the DIVAD variants compared to the unsupervised baselines of Table 4.1 was primarily due to a far better performance of point modeling methods, now even outperforming the sequence modeling ones (0.83 here vs. 0.46 in Table 4.1 for the `Point Average` row, and 0.75 vs. 0.67 for the `Sequence Average` row) (**C8**). This again relates to our previous discussion, indicating that the domain generalization performed by the Dense DIVAD methods was sufficient to turn the *contextual* anomaly components of T3 events into easier-to-detect *point* anomalies, rendering their *collective* components unnecessary. Regarding **T5** (Process Failure) events, the same conclusion can be made as for the unsupervised baselines, with their short and point-based nature favoring the point modeling methods over the sequence modeling ones (**C9**). This advantage of point modeling methods was even more pronounced here, with domain generalization also allowing to turn the few contextual components of T5 events into point anomalies (0.95 over 0.74 peak F1-scores for the DIVAD variants vs. 0.87 over 0.82 for the unsupervised baselines).

Like for our unsupervised baselines, **T6** (Executor Failure) events were by far the **hardest type** to detect (**C10**), indicating domain generalization was not helpful in better separating them from normal data within the test traces. Domain generalization however appeared beneficial to better consistently detect T4 (CPU Contention) events, with an improved `Average` F1-score of 0.72 over 0.60 for the unsupervised baselines (**C11**). The maximum T4 performance could however not be improved (0.75 peak F1-score for Dense DIVAD-G vs. 0.77 for Rec AE).

Sensitivity Analysis: Anomaly Scoring Strategy

Figure 5.10 presents a sensitivity analysis of the anomaly scoring strategy used by our DIVAD methods. It shows the box plots of peak F1-scores achieved by each DIVAD variant and anomaly scoring strategy, with "(P)" indicating the scoring is based on the class encoding *prior* (fixed Gaussian for DIVAD-G, learned Gaussian Mixture for DIVAD-GM), and "(AP)" indicating the scoring is based on the class encoding *aggregated posterior* (estimated as a Gaussian distribution for DIVAD-G, and as a Gaussian Mixture distribution with K components for DIVAD-GM).

As we can see from this figure, sequence modeling DIVAD methods again performed worse than the point modeling variants in both median and maximum peak F1-score no matter the scoring strategy used (**C12**). Like expected, deriving the anomaly scores from an aggregated posterior estimate instead of the prior was significantly beneficial for both DIVAD-G methods (**C13**), which, by relying on a fixed Gaussian prior, are particularly subject to the issue of "holes in the aggregated posterior" discussed in Section 5.2.2. By relying on a *more expressive* and *learned* class encoding prior, DIVAD-GM was less sensitive to the type of scoring strategy used (**C14**), with the scoring based on the prior performing better for the point modeling method, and the one based on the aggregated posterior performing better for the sequence modeling method. This observation is also consistent with our expectations, and motivated our choice of including both scoring strategies into

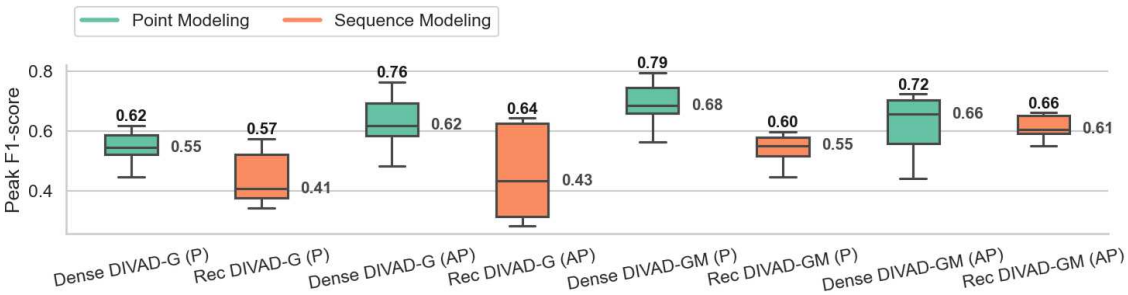


Figure 5.10: Box plots of peak F1-scores achieved by each DIVAD variant and anomaly scoring strategy (class encoding prior (P) vs. aggregated posterior (AP)), colored by modeling strategy (point vs. sequence).

the hyperparameters grid of DIVAD-GM in our study.

5.4 Broader Applicability: Application Server Dataset

This section studies the broader applicability of our DIVAD framework, employing it to detect anomalies in the Application Server Dataset (ASD) [13]. This dataset, collected from a large Internet company, consists of a set of *traces*, each of which recorded the status of a group of services running on a separate *server*, using 19 metrics every five minutes. Although the metrics were anonymized, they for instance relate to the CPU, memory, network or virtual machine of the recorded server. The goal of ASD is to detect the labeled anomaly ranges located at the end of the traces based on the previous data. Its contamination factor (i.e., anomaly ratio) is 4.61%, with minimum, median and maximum anomaly lengths of 3, 18 and 235 data records, respectively. We refer the reader to the original paper for additional details about this dataset.

5.4.1 Experimental Setup and Methods Considered

ASD contains 12 different traces, each corresponding to a server. Its intended usage is to consider a separate instance of anomaly detection model per trace, using the first 30 days of the trace as training and its remaining 15 days as test. In this work, we however use ASD to assess the extent to which our DIVAD framework can learn *server-invariant* normal patterns to detect anomalies in a new, *unseen* test server. As such, our experimental setup considers 11 out of the 12 traces as the training set of a single model instance, and the remaining trace as the test set. Since we do not have information about the similarity between servers, we run 12 separate experiments, each using a given server trace as the test set. For each experiment, we further remove the anomalous records from the training traces, since we assume our methods to be trained on mostly-normal data. For these 12 different runs, we report the performance of TranAD (our best performing unsupervised baseline) and Rec DIVAD-GM (which, this time, outperformed Dense DIVAD-GM in our initial experiments). We consider a window length $L = 20$ for both methods, as well as the same model training and selection methodology as for the Spark Streaming dataset (see Section 4.4.1).

We report the performance of TranAD using the implementation of Tuli et al. [14], and the same preprocessing and hyperparameters as for the Spark Streaming dataset. For Rec DIVAD-GM, we consider each server trace as a separate *domain*, and report its performance using the following preprocessing and hyperparameters:

- A standardization of the input samples.

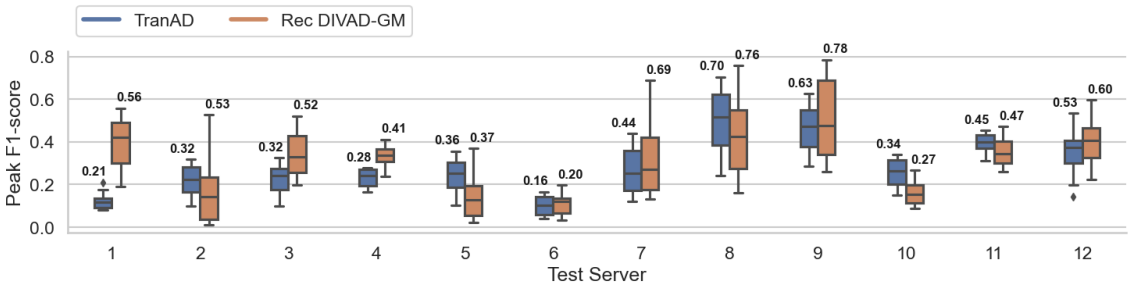


Figure 5.11: Box plots of peak F1-scores achieved by TranAD and Rec DIVAD-GM for ASD, using each server as a test set.

- Each encoder $\text{NN}_{\boldsymbol{\phi}_y}$ and $\text{NN}_{\boldsymbol{\phi}_d}$ with a 1D convolutional layer using 32 filters of size 5, a stride length of 1 and the ReLU activation function, followed by a GRU layer of 32 units using the tanh activation function, and a fully-connected layer to output the encoding parameters.
- The decoder $\text{NN}_{\boldsymbol{\theta}_{yd}}$ defined symmetrically to one encoder as per the design of Figure 4.8.
- A single fully-connected hidden layer of 32 units for the conditional domain prior network $\text{NN}_{\boldsymbol{\theta}_d}$.
- A ReLU activation function followed by a single fully-connected layer of $N_{\text{dom}} = 11$ units (i.e., one per source domain) for the classification head $\text{NN}_{\boldsymbol{\omega}_d}$.
- An encoding dimension of 16, with $K = 8$ Gaussian Mixture components.
- A fixed KL divergence weight $\beta \in \{1, 5\}$.
- A domain classification weight $\alpha_d = 1,000$.
- A mini-batch size $B = 128$.
- An anomaly scoring based on the class encoding aggregated posterior estimate $\hat{q}_{\boldsymbol{\phi}_y}(\mathbf{z}_y)$, fitting a Gaussian Mixture distribution with $K = 8$ components to the training class encodings.

5.4.2 Results and Analyses

Figure 5.11 presents the results of our 12 experiments, showing the box plots of the peak F1-scores achieved by TranAD and Rec DIVAD-GM across their hyperparameter values for each test server. From this figure, we can see that Rec DIVAD-GM outperformed TranAD in maximum peak F1-score for 11 out of 12 test servers (i.e., 92% of the cases), improving the maximum performance by more than 10% for eight of them. However, these results also show that the *median* performance was only improved by Rec DIVAD-GM for 7 out of the 12 possible test servers. This again highlights the sensitivity of our DIVAD framework with respect to the hyperparameters used, and the necessity of properly tuning them to benefit from a performance gain.

Figures 5.12a and 5.12b show the KDE plots of the anomaly scores assigned by the best-performing TranAD and Rec DIVAD-GM methods to training normal, test normal and test anomalous records when using server 1 as a test set (i.e., the setup for which Rec DIVAD-GM improved the performance the most, by 167%). From Figure 5.12a, we can see the low performance of TranAD was primarily to the lower mode of its distribution of anomaly scores assigned to anomalies, which had a significant overlap, and thus were considered "similarly abnormal", to some of the test normal data. As we can see in Figure 5.12b, Rec DIVAD-GM was able to alleviate this issue, producing much less overlap between this lower mode and the rest of test normal data, which resulted in the performance gain.

An important aspect to note about these experiments is that, since we do not have

(a) TranAD. (b) Rec DIVAD-GM.

Figure 5.12: Kernel Density Estimate (KDE) plots of the anomaly scores assigned by TranAD and Rec DIVAD-GM and to the training normal, test normal and test anomalous records of ASD, using server 1 as a test set.

information about the nature of the services running on the servers, it is difficult to know *a priori* whether learning normal patterns that are shared across the training servers will be useful to detect anomalies in a new one. From this study, we could see that this was mostly the case, with the clearest example being for test server 1. We however observed some disparities in the results, hinting at different properties for the server behaviors. As an example, learning from the rest of the servers seemed sufficient to detect anomalies in test server 8, even without resorting to domain generalization, with high maximum F1-scores of 0.70 and 0.76 achieved by TranAD and Rec DIVAD-GM, respectively. In contrast, learning patterns that generalized to test server 6 was found to be more challenging, whether using domain generalization or not, with much lower maximum F1-scores of 0.16 and 0.20 obtained for the respective methods. In other terms, and at least from these experiments, (i) services running on server 1 were found different from those running on the other servers, but still possible to capture based on shared underlying factors, (ii) services running on server 8 were found "similar" to (at least some) services running on the other servers, while (iii) services running on server 6 were found different from the others, both in "direct" and "underlying" behaviors.

5.5 Summary and Conclusions

In this chapter, we started by formally characterizing the *normal behavior shift* challenge identified in Chapter 4, relating it to the concept of **domain shift**. We then sought to address this challenge explicitly, by proposing **DIVAD**, a domain generalization method decomposing the observed variable into *domain-specific* and *domain-invariant* encodings, and defining anomalies as samples that deviate from the training distribution of domain-invariant encodings only. After elaborating on the model design, model training and possible anomaly scoring strategies of DIVAD, we applied multiple of its variants against our Exathlon benchmark and experimental setup. We presented our results and analyses from this study, which led to **14 conclusions**, summarized, reorganized and grouped by similarity below:

- **C1, C3, C4.** Our best-performing DIVAD variants significantly improved the detection results of our unsupervised baselines. This improvement was directly attributable to their explicit domain generalization, which addressed most of the normal behavior shift challenge identified in the last chapter. The high variance in the results obtained for a given DIVAD method however showed their particular sensitivity to the hyperparameter used, and the necessity of properly tuning them to benefit from a performance gain in practice.
- **C2.** Within our DIVAD variants, considering a learned Gaussian Mixture prior (DIVAD-GM) over a fixed Gaussian prior (DIVAD-G) was always found beneficial.

- **C5, C6.** For our dataset and experimental setup, using point modeling over sequence modeling DIVAD variants was both *sufficient* and *necessary* to outperform our unsupervised baselines. Point modeling variants being sufficient was expected due to most event types predominantly inducing contextual anomalies, which could be turned into easier-to-detect point anomalies when viewed in a domain-invariant space. Sequence modeling variants performing worse was mainly explained by (i) their domain-invariant mapping removing some relevant anomaly signals from the data, due to a lack of incentive to preserve them, and (ii) the more challenging nature of learning domain-invariant shapes within and across $M = 237$ time series for this dataset.
- **C7, C8, C9.** Within test traces, the easiest types of events to detect across the DIVAD variants were T2 (Bursty Input Until Crash), T3 (Stalled Input) and T5 (Driver Failure). Like for the unsupervised baselines, the point modeling variants outperformed the sequence modeling ones in detecting T5 events. The worse detection of T1 events compared to the unsupervised baselines was primarily due to the recurrent DIVAD-G's domain-invariant mapping removing important T1 signals from the data, while the better detection of T3 events was mainly due to a better ability of point modeling variants to detect them.
- **C10, C11.** Within test traces, the hardest type of event to detect was T6 (Executor Failure), similar to the unsupervised baselines. This indicates that the domain generalization performed by the DIVAD methods did not help better distinguish T6 events from normal data within a test trace. Domain generalization however appeared beneficial to consistently detect T4 (CPU Contention) events, although the maximum performance achieved for these events could not be improved.
- **C12, C13, C14.** Sequence modeling DIVAD methods performed worse than point modeling variants in both median and maximum peak F1-score no matter the scoring strategy used. Like expected, deriving the anomaly scores from an aggregated posterior estimate instead of the prior significantly improved the performance of both DIVAD-G methods. By relying on a more expressive and learned class encoding prior, DIVAD-GM was less sensitive to the scoring strategy used.

This chapter ended with a study of the broader applicability of our DIVAD framework, employing it to detect anomalies in a server trace from the Application Server Dataset (ASD) given the traces from the 11 other servers as training. This study revealed that our DIVAD method could outperform TranAD in most of the test cases, showing that its explicit domain generalization could also be useful beyond our Spark Streaming dataset.

Although the study of Chapter 4 identified (**L1**) *the vulnerability to normal behavior shift from training to test data* as the main limitation of our unsupervised baselines. It also revealed their performance was significantly hindered by (**L2**) *their production of false negatives for the hardest anomalies given our large number of features* and (**L3**) *production of false positives for normal but "noisy", minority patterns in test data*. In the next chapter, our goal will be to better address all three of these limitations **L1-3** in a setting of weakly-supervised learning.

6 Prior Knowledge through Weak Supervision

In this chapter, we introduce *prior knowledge* about the relevant anomalies to detect through the setting of "learning with incomplete supervision", which we simply refer to as **weakly-supervised learning**. We start by formalizing this new setting with a revised problem statement in Section 6.1. We then present **CEADAL** in Section 6.2, a method relying on *contrastive learning* to learn a compact representation of normal data, away from the few labeled anomalies. We also present an alternative method based on the triplet loss called **TEADAL**. In Section 6.3, we apply CEADAL and TEADAL against the Exathlon benchmark and our experimental setup extended with the few training anomalies, comparing them with classification and weakly-supervised anomaly detection methods. We derive 18 conclusions from this study, highlighted as **C1-18**, and show that the compact representation of data in latent space learned by CEADAL can implicitly address the (**L1**) *vulnerability to normal behavior shift from training to test data* identified for the unsupervised baselines in Chapter 4, while also better addressing their (**L2**) *production of false negatives for the hardest anomalies given our large number of features* and (**L3**) *production of false positives for normal but "noisy", minority patterns in test data*.

6.1 Revised Problem Statement

This section formalizes the anomaly detection problem we want to solve in this weakly-supervised setting.

We consider N_1 training sequences and N_2 test sequences:

$$\mathcal{S}_{\text{train}} = (\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(N_1)}) \ , \ \mathcal{S}_{\text{test}} = (\boldsymbol{S}^{(N_1+1)}, \ldots, \boldsymbol{S}^{(N_1+N_2)}),$$

where each $\boldsymbol{S}^{(i)}$ consists of T ordered data records of dimension M, with corresponding sequences of *anomaly labels*:

$$\mathcal{Y}_{\text{train}} = \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(N_1)}\} \quad , \quad \mathcal{Y}_{\text{test}} = \{\boldsymbol{y}^{(N_1+1)}, \ldots, \boldsymbol{y}^{(N_1+N_2)}\},$$

with $\boldsymbol{y}^{(i)} \in \{0, 1\}^T$, such that:

$$\begin{cases} y_t^{(i)} = 1 & \text{if the record at index } t \text{ in sequence } i \text{ is } \textit{anomalous}, \\ y_t^{(i)} = 0 & \text{otherwise (i.e., the record is } \textit{normal}). \end{cases}$$

The rest of the problem statement is the same as described in Section 4.1, with the record scoring function $g : \mathbb{R}^{T \times M} \rightarrow \mathbb{R}^T$ now trained on $\mathcal{S}_{\text{train}}$ and $\mathcal{Y}_{\text{train}}$, instead of just $\mathcal{S}_{\text{train}}$ in the unsupervised setting. Like mentioned in Chapter 5, we use $\boldsymbol{x}$ and y to refer to an input window and its corresponding label in the following, instead of $\boldsymbol{X}$ and $\tilde{y}$ used in Section 3.3.3, to simplify the notation.

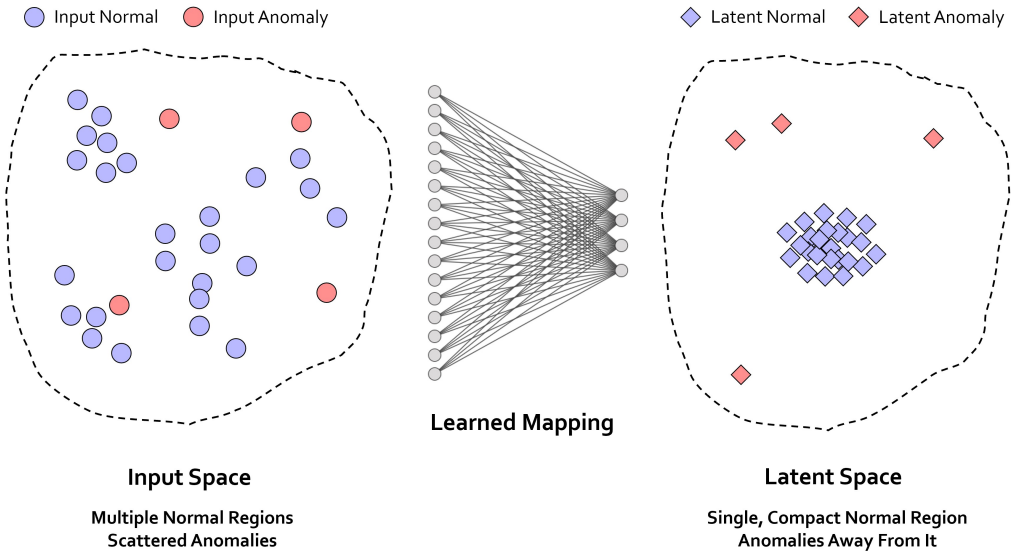


Figure 6.1: Our goal is to map the normal data samples to a single compact region, and anomalies away from it.

6.2 Contrastive Encoder for Anomaly Detection with a Few Anomaly Labels

This section presents Contrastive Encoder for Anomaly Detection with a Few Anomaly Labels (CEADAL), a method relying on contrastive learning to construct a latent mapping where normal data samples get mapped to a compact region, away from the anomalous samples. We start by motivating and describing this framework in Section 6.2.1. We then detail the contrastive pair mining and anomaly scoring strategy of CEADAL in Sections 6.2.2 and 6.2.3, respectively. We end this section by presenting an alternative of CEADAL based on the more recent and less-constrained triplet loss, called Triplet Encoder for Anomaly Detection with a Few Anomaly Labels (TEADAL).

6.2.1 Contrastive Learning Framework

To address the challenges identified in Chapter 4, we propose to leverage the few labeled anomalies to construct a *latent space mapping* such that:

- Normal data samples get mapped to a single, compact region.
- This normal region is learned *in contrast* to the few anomaly representatives, which should get mapped away from it.

We illustrate this motivation in Figure 6.1[1]. To construct this mapping, we present "Contrastive Encoder for Anomaly Detection with a Few Anomaly Labels", or CEADAL. This method relies on *contrastive learning* [22], adapted for anomaly detection by extracting normal-normal pairs and normal-anomaly pairs from the training samples, referred to as *concordant* and *discordant* pairs, respectively. After extracting these pairs, the mapping is learned by training a Siamese neural network to *encode* the samples from concordant pairs "near each other", and the samples from discordant pairs "far apart" in a *latent* space, as illustrated in Figure 6.2.

[1]The neural network diagrams of this chapter were created using [128].

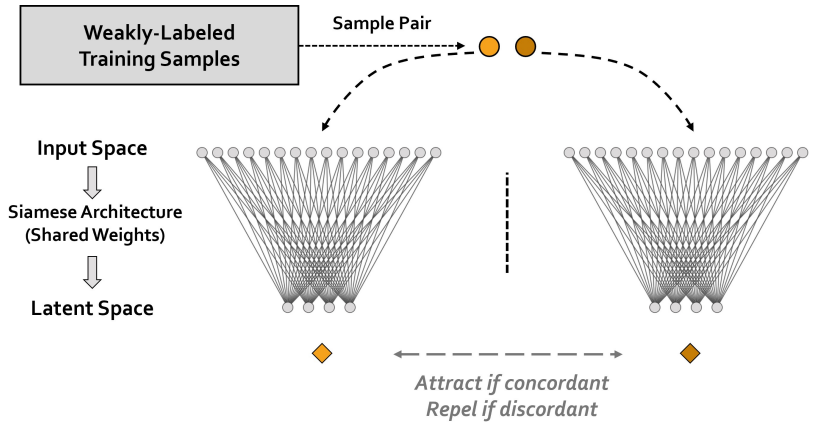


Figure 6.2: Training procedure: update the weights of a Siamese neural network to attract the members of concordant pairs and repel the members of discordant pairs.

Formally, our goal is to learn the parameters $\boldsymbol{\theta}$ of a mapping $e_{\boldsymbol{\theta}} : \mathbb{R}^M \to \mathbb{R}^{M'}$, called *contrastive encoder*, that minimize the following contrastive loss [22]:

$$\mathcal{L}(\mathcal{P}; \boldsymbol{\theta}) = \frac{1}{|\mathcal{P}|} \sum_{(\boldsymbol{x}_1, \boldsymbol{x}_2, y) \in \mathcal{P}} L_C(\boldsymbol{x}_1, \boldsymbol{x}_2, y; \boldsymbol{\theta}),$$

where:

$$L_C(\boldsymbol{x}_1, \boldsymbol{x}_2, y; \boldsymbol{\theta}) = (1-y)\|e_{\boldsymbol{\theta}}(\boldsymbol{x}_1) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_2)\|_2^2 + y \max\left\{0, m - \|e_{\boldsymbol{\theta}}(\boldsymbol{x}_1) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_2)\|_2\right\}^2,$$

$\mathcal{P}$ is the set of labeled concordant and discordant pairs extracted from the training samples, with $y = 0$ for concordant pairs and $y = 1$ for discordant pairs, and $m \in \mathbb{R}_+^*$ is a margin hyperparameter, after which increasing the distance for discordant pairs is no longer rewarded.

6.2.2 Pair Mining Strategy

An important factor in successfully training the contrastive encoder presented in Section 6.2.1 is the strategy we adopt to extract concordant and discordant pairs from the training samples. Some pairs can indeed turn out to be "easy" or "hard" for the model at any time during its training process, with both *easy concordant pairs* (i.e., normal samples already mapped very close to each other in latent space) and *easy discordant pairs* (normal and anomaly samples already mapped farther than m in latent space) contributing (close to) nothing to the contrastive loss, which reduces the useful information provided.

To address this issue, we adopt a strategy similar to "batch hard" triplet mining [23], and train the contrastive encoder only on the "hardest" concordant and discordant pairs in every given mini-batch. Specifically, at every epoch during training, we constitute mini-batches of (even) size B, each *balanced* with respect to the sample class ("normal" vs. "anomaly") by oversampling the anomalous examples. For every mini-batch, we then consider the data samples encoded by the model in latent space (i.e., perform a model forward pass), and pair each of the $B/2$ normal samples to (i) its furthest other normal sample and (ii) its closest anomalous sample, constituting a set of B "hardest" concordant and discordant pairs. We finally use these pairs to compute the contrastive loss and update the weights of the network. This overall process is illustrated in Figure 6.3.

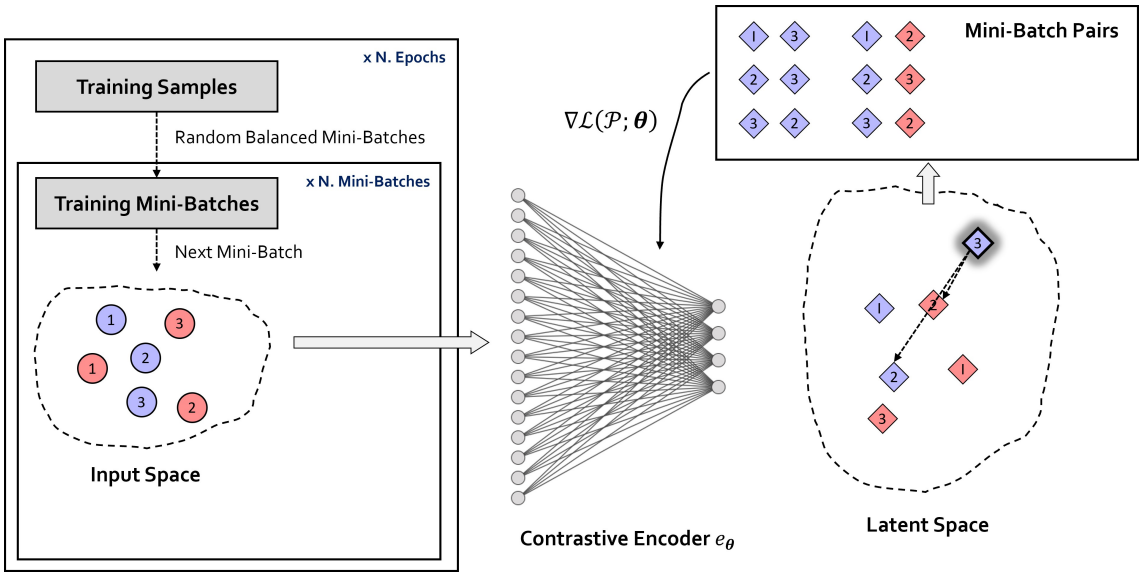


Figure 6.3: "Batch hard" pair mining strategy for the contrastive encoder training: normal samples 1, 2 and 3 get paired with their closest normal and furthest anomalous sample in latent space.

6.2.3 Anomaly Scoring

After training the contrastive encoder, we expect normal data samples to get mapped to a single, compact, spherical region in latent space, away from anomalies. In practice, this spherical region is however likely to end up imperfect, resembling more an approximate ellipsoid. This motivates estimating the distribution of the training normal samples in latent space more generally as a multivariate Gaussian, and defining the **anomaly score** of a test sample $\boldsymbol{x}$ as the *squared Mahalanobis distance* from this distribution:

$$g_W(\boldsymbol{x}) := d^2_{\text{Maha}}(e_{\boldsymbol{\theta}}(\boldsymbol{x}); \boldsymbol{c}, \boldsymbol{\Sigma}) = (e_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{c})^T \boldsymbol{\Sigma}^{-1} (e_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{c}),$$

with $\boldsymbol{c} \in \mathbb{R}^{M'}$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{M' \times M'}$ the empirical mean and covariance matrix of the training normal data in latent space, respectively. Relying on the squared Euclidean distance to derive our anomaly scores directly would indeed not be optimal for an ellipsoidal normal region, as lower distance thresholds would be relevant for lower-variance axes. Using the Mahalanobis distance definition above directly addresses this issue, as it amounts to first rescaling each principal axis of the latent normal cluster to have unit variance, and then applying the squared Euclidean distance for anomaly scoring.

As we can see, the framework of CEADAL still focuses mainly on capturing the *normal behavior* of the data, by modeling the distribution of normal data samples in latent space to derive its anomaly scores. As such, we can expect it to be more robust to class imbalance and labels scarcity than traditional classifiers. By contrasting the normal region learned with respect to a few labeled anomalies, this framework however injects *prior knowledge* about the relevant anomalies to detect. For this reason, we can expect CEADAL to outperform unsupervised methods in (i) *detecting the impacts of harder, more "subtle" anomalous events on high-dimensional feature sets*, but also (ii) making *real anomalies stand out from normal but "noisy", minority patterns* in the data, hence addressing the corresponding challenges of Chapter 4. Besides, by enforcing a *similar* representation of the normal data samples *no matter the context they come from*, our contrastive framework embeds a context generalization component, which implicitly addresses our remaining *normal behavior shift* challenge, as further shown in our experimental study.

6.2.4 Triplet Loss Alternative

To study the impact of the implicit context generalization induced by CEADAL's contrastive loss, we propose an alternative encoder based on the more recent and less-constrained *triplet loss* [23], called "Triplet Encoder for Anomaly Detection with a Few Anomaly Labels" (TEADAL). Rather than enforcing normal-normal distances to approach zero, this framework simply requires them to be smaller than normal-anomaly distances by a margin $m \in \mathbb{R}_+^*$, minimizing the following triplet loss [23]:

$$\mathcal{L}(\mathcal{T}; \boldsymbol{\theta}) = \frac{1}{|\mathcal{T}|} \sum_{(\boldsymbol{x}_a, \boldsymbol{x}_p, \boldsymbol{x}_n) \in \mathcal{T}} L_T(\boldsymbol{x}_a, \boldsymbol{x}_p, \boldsymbol{x}_n; \boldsymbol{\theta}),$$

where:

$$L_T(\boldsymbol{x}_a, \boldsymbol{x}_p, \boldsymbol{x}_n; \boldsymbol{\theta}) = \max\left\{ \|e_{\boldsymbol{\theta}}(\boldsymbol{x}_a) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_p)\|_2^2 - \|e_{\boldsymbol{\theta}}(\boldsymbol{x}_a) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\|_2^2 + m, 0 \right\},$$

and $\mathcal{T}$ is a set of *triplets* extracted from the training samples, where both the *anchor* $\boldsymbol{x}_a$ and *positive sample* $\boldsymbol{x}_p$ are normal, and the *negative sample* $\boldsymbol{x}_n$ is an anomaly. The goal is therefore to *maximize* $\|e_{\boldsymbol{\theta}}(\boldsymbol{x}_a) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\|_2^2 - \|e_{\boldsymbol{\theta}}(\boldsymbol{x}_a) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_p)\|_2^2$ (i.e., make the squared normal-anomaly distance larger than the squared normal-normal distance) until we have $\|e_{\boldsymbol{\theta}}(\boldsymbol{x}_a) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\|_2^2 \geq \|e_{\boldsymbol{\theta}}(\boldsymbol{x}_a) - e_{\boldsymbol{\theta}}(\boldsymbol{x}_p)\|_2^2 + m$ (i.e., until a margin m separates the two squared distances).

The training strategy for this triplet encoder is then the same as for the contrastive encoder, except we consider a single triplet where we previously considered a concordant and a discordant pair. Like in [23] and CEADAL, we only train the encoder on the "hardest" triplets in every given mini-batch. Also like in [23], we stabilize training by further *L2-normalizing* the encoder outputs. In this context, minimizing the squared Euclidean distance between two latent samples is equivalent to maximizing their cosine similarity. We therefore define the **anomaly score** of a test sample $\boldsymbol{x}$ as its *negative cosine similarity* to the training normal centroid in latent space:

$$\boxed{g_W(\boldsymbol{x}) := -S_{\cos}(e_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{c}) = -\boldsymbol{c}^T e_{\boldsymbol{\theta}}(\boldsymbol{x}),}$$

where $\boldsymbol{c} \in \mathbb{R}^{M'}$ is the empirical mean of the training normal data in latent space.

By relying on the triplet loss over the more constrained contrastive loss, we can expect TEADAL to learn a richer representation of normal data, allowing the latent normal region to exhibit more *intra-class variance*. Enabling such flexibility is especially relevant in the general anomaly detection setting, where a single "normal" class is typically assigned to refer to a possibly wide range of different normal behaviors. As our experiments will show, this flexibility however acts as a hindrance for our specific use case and setup, by removing the heavier compactness constraint of the contrastive loss for the latent normal region, and thus its implicit context generalization component.

6.3 Experiments

This section applies the CEADAL and TEADAL methods introduced in Section 6.2 against the Exathlon benchmark and dataset, and compares them to other methods from the literature that rely on class labels. We start by presenting our revised experimental setup for this weakly-supervised setting. We then describe the compared methods and their grid of hyperparameter values. We finally present and analyze the results obtained, deriving 18 conclusions, labeled **C1-18**, showing in particular that our CEADAL method could (at least partly) address the three main challenges identified in Chapter 4.

6.3.1 Revised Experimental Setup

For this weakly-supervised study, we no longer remove the labeled anomaly ranges from our training and validation sequences, resulting in the following anomaly ranges in the *training set* of the compared methods:

- 8 Bursty Input (**T1**) ranges.
- 2 Bursty Input Until Crash (**T2**) ranges.
- 2 Stalled Input (**T3**) ranges.
- 6 CPU Contention (**T4**) ranges.
- 2 Driver Failure (**T5**) ranges.
- 2 Executor Failure (**T6**) ranges.

And the following anomaly ranges in their *validation set*:

- 6 Bursty Input (**T1**) ranges.
- 2 Stalled Input (**T3**) ranges.
- 4 CPU Contention (**T4**) ranges.
- 1 Driver Failure (**T5**) ranges.
- 2 Executor Failure (**T6**) ranges.

Like mentioned in Section 4.4.1, T2 traces were indeed all used as training because of their shorter lengths. For a window length $L = 1$, and after the window balancing described in Section 4.2.5, this setup corresponds to a **training contamination factor** of **3.36%**, (0.66%, 1.74%, 0.22%, 0.64%, 0.01% and 0.07% for T1 to T6, respectively) and a **validation contamination factor** of **3.44%** (0.74%, 0.30%, 0.78%, 0.01% and 1.62% for T1, T3, T4, T5 and T6, respectively). For window lengths $L > 1$, we only keep the anomalous windows that either (i) were 100% abnormal or (ii) contained all of an anomaly range (to handle cases where anomaly ranges were shorter than our window length). After this process and window balancing for a window length $L = 20$, this setup corresponds to a **training contamination factor** of **3.44%** (0.69%, 1.76%, 0.22%, 0.67%, 0.02% and 0.08% for T1 to T6, respectively), and a **validation contamination factor** of **3.48%** (0.74%, 0.30%, 0.80%, 0.01% and 1.62% for T1, T3, T4, T5 and T6, respectively).

The rest of the experimental setup is the same as for our unsupervised setting (which makes the methods from both settings comparable).

6.3.2 Compared Methods and Hyperparameters

As described in Section 6.2, the objective of both our CEADAL and TEADAL methods is to learn a transformation to a latent space where normal and anomalous samples are clearly separated, making them operate in an *anomaly feature representation learning* framework [15]. Besides CEADAL and TEADAL, we include another, state-of-the-art, anomaly feature representation learning method called Deep SAD [62] in our comparative study, along with some relevant classification methods. On the whole, we consider the following *encoding* and *classification* methods:

- **Dense CEADAL** and Convolutional CEADAL (**Conv CEADAL**), as point modeling and sequence modeling CEADAL variants, respectively.
- **Dense TEADAL** and Convolutional TEADAL (**Conv TEADAL**), as point modeling and sequence modeling TEADAL variants, respectively.
- Dense Deep SAD [62] (**Dense DSAD**) and Convolutional Deep SAD (**Conv DSAD**), as point modeling and sequence modeling Deep SAD variants, respectively, and Deep SAD a relevant anomaly feature representation learning method.

- **XGBoost** [129], as a popular, shallow, point modeling method used for classification.
- **Dense DIVA** [71] and Convolutional DIVA (**Conv DIVA**), as point modeling and sequence modeling DIVA variants, respectively, and DIVA a relevant domain generalization method designed for classification.

For all the deep learning methods, we adopt the same model training and selection strategy as described in Section 4.4.1. For the sequence modeling methods, we consider a window length $L = 20$, like for the unsupervised sequence modeling methods of Chapters 4 and 5.

We report the performance of Dense CEADAL using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- A single hidden layer of 200 units.
- An latent mapping (i.e., encoding) dimension in $\{16, 64\}$.
- The ReLU activation function for all the layers except the encoding.
- A mini-batch size $B = 300$. Using large mini-batches is indeed advised when relying on an online, "batch hard" pair mining strategy [23].
- A contrastive loss margin $m = 10$.

When constituting the mini-batches of both CEADAL and TEADAL, we further consider the anomalous samples *balanced* by event type, resulting in $B/2 = 150$ normal samples and $B/(2 \cdot 6) = 25$ anomalous samples of each type per batch. This amounts to assuming that all the event types were equally represented in the data. None of the methods however relied on the event type information of the anomalous samples; they only considered the *binary* detection problem described in Section 6.1.

We report the performance of Conv CEADAL using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- Two 1D convolutional layers, each using 32 filters of size 5, a stride length of 1, batch normalization and the ReLU activation function (with a 1D max-pooling layer of window and stride lengths 2 after the first convolutional layer), followed by a fully-connected layer to output the encoding.
- An encoding dimension in $\{64, 128\}$, with no activation function for the encoding layer.
- A mini-batch size $B = 300$.
- A contrastive loss margin $m = 10$.

We report the performance of Dense TEADAL and Conv TEADAL using the same preprocessing and hyperparameters as Dense CEADAL and Conv CEADAL, respectively, with a triplet loss margin $m = 0.2$ in place of the contrastive loss margin. For both TEADAL variants, we however consider a lower grid of learning rate values $\eta \in \{1e{-}6, 3e{-}6, 1e{-}5, 3e{-}6\}$ than for other deep learning methods, as we observed a higher tendency of TEADAL's validation losses to diverge using the regular grid in our experiments.

Deep SAD [62] (for "Deep Semi-supervised Anomaly Detection") is a follow-up work of Deep SVDD [54], improving on its latent mapping by imposing that anomalous samples get sent far from the centroid of the normal hypersphere. Like CEADAL, this method aims to learn similar representations of normal data samples no matter the context they come from, inducing a *proximity constraint* between them that is likely to benefit context generalization. For this study, we use the Deep SAD implementation of Ruff et al. [62], including their initialization of the encoder weights from a pretrained autoencoder model. We report the performance of Dense DSAD with the following preprocessing and hyperparameters (using the same values as the original implementation for those not mentioned):

- A standardization of the input samples (we also tried the original paper's strategy of normalizing the inputs and using a sigmoid activation function for the output layer, but this did not lead to a better performance).
- A single hidden layer of 200 units for the encoder and the decoder (with the decoder only being used for pretraining).
- An encoding dimension in $\{16, 64\}$.
- The Leaky ReLU activation function with a negative slope coefficient $\alpha = 0.01$ for all the layers except the encoding and decoder output (like in the original implementation).
- A mini-batch size $B = 200$ (like in the original implementation).
- A pretraining phase of 150 epochs, followed by a training phase of 150 epochs (which makes the same total number of 300 epochs as the other methods).
- The same learning rate and optimization strategy for the pretraining and training phases.
- The same grid of learning rate values as the other methods, but dividing the learning rate by 10 after 50 epochs (like in the original implementation).
- A weight decay coefficient of 1e−6 (like in the original implementation).

We report the performance of Conv DSAD with the following preprocessing and hyperparameters (using the same values as the original implementation for those not mentioned):

- A standardization of the input samples.
- An encoder with two 1D convolutional layers, each using 32 filters of size 5, a stride length of 1, batch normalization and the Leaky ReLU activation function (with a negative slope coefficient $\alpha = 0.01$, and a 1D max-pooling layer of window and stride lengths 2 after the first convolutional layer), followed by a fully-connected layer to output the encoding.
- A pretraining decoder defined symmetrically to the encoder as per the design of Figure 4.8, with a 1D upsampling layer to invert the encoder's 1D max-pooling layer.
- An encoding dimension in $\{64, 128\}$, with no activation function for the encoding layer.
- A mini-batch size $B = 200$ (like in the original implementation).
- A pretraining phase of 150 epochs, followed by a training phase of 150 epochs.
- The same learning rate and optimization strategy for the pretraining and training phases.
- The same grid of learning rate values as the other methods, but dividing the learning rate by 10 after 50 epochs (like in the original implementation).
- A weight decay coefficient of 1e−6 (like in the original implementation).

XGBoost [129] (for "eXtreme Gradient Boosting") is an optimized, distributed implementation of gradient boosted trees, building an ensemble of decision trees sequentially to perform a supervised learning task, with each tree correcting the errors made by the previous ones. For this study, we apply XGBoost for anomaly detection by training it for binary classification (using a logistic regression objective), and defining the anomaly score of a test sample as its predicted probability to belong to the anomalous class. We report the performance of XGBoost with the hyperparameter values that maximized the validation F1-score after 100, 200, 500, 1,000 and 2,000 trials of Bayesian hyperparameter optimization over the following hyperparameter space (using the default values of XGBoost 2.0.3 for the ones not mentioned):

- A number of boosting rounds in $\{50, 100, 200\}$.

- A maximum tree depth in $[1\,..\,12]$.
- A minimum "child weight" in $[1, 3, 5, 10, 20, 100, 500]$.
- A subsampling ratio in $[0.5, 1.0]$.
- A boosting learning rate between 1e$-$4 and 0.5, using the "log sampling" strategy of KerasTuner [130] (i.e., uniformly sampling $\eta \in [0.0, 1.0)$ and setting the boosting learning rate to $1\text{e}{-4} \cdot (0.5/1\text{e}{-4})^{\eta}$).
- A minimum required loss reduction $\gamma = 0.0$.
- A maximum delta step in $[0, 1, 5]$.
- A choice between balancing the data samples (i.e., setting the balancing of positive and negative weights to the number of normal samples over the number of anomalous samples) or not (i.e., setting the balancing of positive and negative weights to 1.0).

DIVA [71] (for "Domain-Invariant Variational Autoencoders") is a domain generalization method proposed for image classification, which we adapted for unsupervised anomaly detection in Chapter 5 to construct our DIVAD method. Its generative model assumes the observed variable $\mathbf{x}$ is generated from independent latent factors $\mathbf{z}_y$, $\mathbf{z}_d$ and $\mathbf{z}_x$, where $\mathbf{z}_y$ represents its class-specific component, conditioned on its class y, $\mathbf{z}_d$ represents its domain-specific component, conditioned on its domain d, and $\mathbf{z}_x$ represents its residual component. DIVA has a similar architecture to DIVAD, with the main differences of:

1. Considering an additional encoder $\text{NN}_{\boldsymbol{\phi}_x}(\mathbf{x})$, outputting the parameters of the variational posterior $q_{\boldsymbol{\phi}_x}(\mathbf{z}_x|\mathbf{x})$.
2. Considering the three latent encodings as input to the decoder $\text{NN}_{\boldsymbol{\theta}}(\mathbf{z}_d, \mathbf{z}_y, \mathbf{z}_x)$, outputting the parameters of the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_d, \mathbf{z}_y, \mathbf{z}_x)$.
3. Using two conditional Gaussian priors $p_{\boldsymbol{\theta}_y}(\mathbf{z}_y|\text{y})$ and $p_{\boldsymbol{\theta}_d}(\mathbf{z}_d|\text{d})$, with corresponding networks $\text{NN}_{\boldsymbol{\theta}_y}(\text{y})$ and $\text{NN}_{\boldsymbol{\theta}_d}(\text{d})$, along with a fixed standard Gaussian prior $p(\boldsymbol{z}_x)$.
4. Using two classification heads $\text{NN}_{\boldsymbol{\omega}_y}(\mathbf{z}_y)$ and $\text{NN}_{\boldsymbol{\omega}_d}(\mathbf{z}_d)$, outputting the parameters of $q_{\boldsymbol{\omega}_y}(\text{y}|\mathbf{z}_y)$ and $q_{\boldsymbol{\omega}_d}(\text{d}|\mathbf{z}_d)$, respectively, with corresponding classification loss weights α_y and α_d in the total loss.

We refer the reader to the original paper for additional details about this method. For this study, we apply DIVA for anomaly detection by defining the anomaly score of a test sample $\boldsymbol{x}$ as its predicted probability of being anomalous from $q_{\boldsymbol{\omega}_y}(\text{y}|\boldsymbol{z}_y)$, with $\boldsymbol{z}_y \sim q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x})$. We consider DIVA without its residual latent factor $\mathbf{z}_x$, as including it tended to result in worse performance in our experiments. Like for DIVAD, we train DIVA with *domains* defined as *trace contexts* (i.e., the Spark settings and input rate used for the application runs), resulting in the same 22 source domains and 11 target domains as in Chapter 5. We further balance the normal and anomaly classes in the data, by oversampling the anomalies to match the cardinality of normal samples for both the training and validation sets (with the anomalous samples also balanced by event type within the anomaly class). We report the performance of Dense DIVA using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- A single hidden layer of 200 units for the encoders $\text{NN}_{\boldsymbol{\phi}_y}$ and $\text{NN}_{\boldsymbol{\phi}_d}$ as well as for the decoder $\text{NN}_{\boldsymbol{\theta}_{yd}}$.
- A single fully-connected hidden layer of 64 units for the conditional class prior network $\text{NN}_{\boldsymbol{\theta}_y}$ and the conditional domain prior network $\text{NN}_{\boldsymbol{\theta}_d}$.
- A ReLU activation function followed by a single fully-connected layer of $N_{\text{dom}} = 22$ units (i.e., one per source domain) for the classification heads $\text{NN}_{\boldsymbol{\omega}_y}$ and $\text{NN}_{\boldsymbol{\omega}_d}$.
- The ReLU activation function for all the layers except the encoding and output layers.
- An encoding dimension in $\{16, 64\}$.

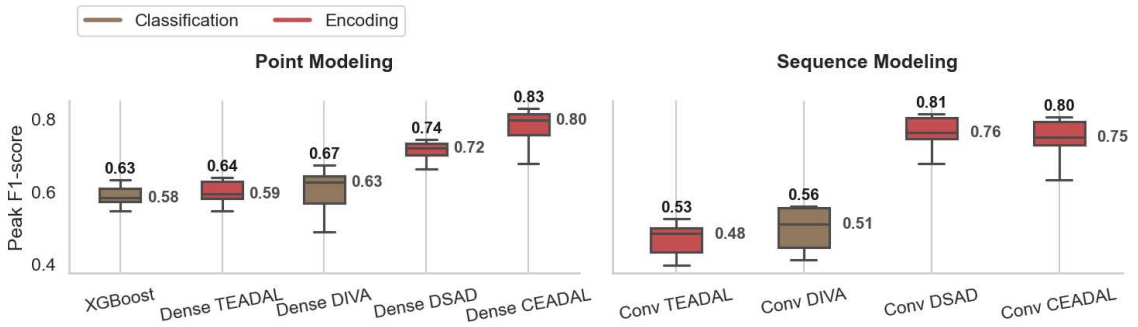


Figure 6.4: Box plots of peak F1-scores achieved by the encoding and classification methods, separated by modeling strategy (point vs. sequence) and colored by method category (encoding vs. classification).

- A fixed KL divergence weight $\beta \in \{1, 5\}$. We also tried using the linear scheduling strategy of the original implementation, but this did not lead to a better performance.
- Weights $\alpha_y = 100,000$ and $\alpha_d = 100,000$ for the class and domain classification losses, respectively. We also tried different weight values, as well as the CoV-Weighting strategy proposed by Groenendijk et al. [127] to automatically balance the three loss terms, but this did not lead to a better performance.
- A mini-batch size $B = 128$ (like in the original implementation).

We report the performance Conv DIVA using the following preprocessing and hyperparameters:

- A standardization of the input samples.
- Each encoder $\text{NN}_{\boldsymbol{\phi}_y}$ and $\text{NN}_{\boldsymbol{\phi}_d}$ with two 1D convolutional layers using 32 filters of size 5, a stride length of 1 for the first layer and 2 for the second layer, and the ReLU activation function, followed by a fully-connected layer to output the encoding parameters.
- The decoder $\text{NN}_{\boldsymbol{\theta}_{yd}}$ defined symmetrically to one encoder as per the design of Figure 4.8.
- The same architectures as Dense DIVA for the conditional class and domain prior networks $\text{NN}_{\boldsymbol{\theta}_y}$ and $\text{NN}_{\boldsymbol{\theta}_d}$, as well as for the classification heads $\text{NN}_{\boldsymbol{\omega}_y}$ and $\text{NN}_{\boldsymbol{\omega}_d}$.
- An encoding dimension of 32.
- A fixed KL divergence weight $\beta \in \{1, 5\}$.
- Weights $\alpha_y = 100,000$ and $\alpha_d = 100,000$ for the class and domain classification losses, respectively.
- A mini-batch size $B = 128$ (like in the original implementation).

6.3.3 Results and Analyses

Figure 6.4 shows the box plots of the peak F1-scores achieved by the encoding and classification methods across their hyperparameter values. It separates point from sequence modeling methods in two different subplots with a shared y-axis, and shows boxes colored based on the method category (encoding vs. classification). Table 6.1 follows the format described in Section 4.5.1, considering the methods with their best-performing hyperparameters (i.e., maximum peak F1-score in Figure 6.4), and reporting their average peak F1-score within each test trace for each type of event.

Method	Tr-T1	Tr-T2	Tr-T3	Tr-T4	Tr-T5	Tr-T6
XGBoost [129]	**0.97**	**1.00**	0.82	0.61	**0.18**	0.46
Dense TEADAL	0.96	0.96	0.59	0.67	0.73	0.36
Dense DIVA [71]	0.90	0.86	0.56	0.57	**0.11**	0.30
Dense DSAD [62]	**0.97**	**1.00**	0.88	0.79	**0.97**	0.54
Dense CEADAL	0.96	**0.98**	**0.89**	**0.85**	**0.95**	**0.56**
Conv TEADAL	0.17	0.93	0.15	0.54	0.94	0.34
Conv DIVA [71]	0.87	0.98	0.56	0.40	**0.31**	0.32
Conv DSAD [62]	0.98	0.98	**0.91**	**0.86**	0.89	**0.61**
Conv CEADAL	**0.99**	0.97	**0.89**	**0.81**	**0.96**	**0.57**

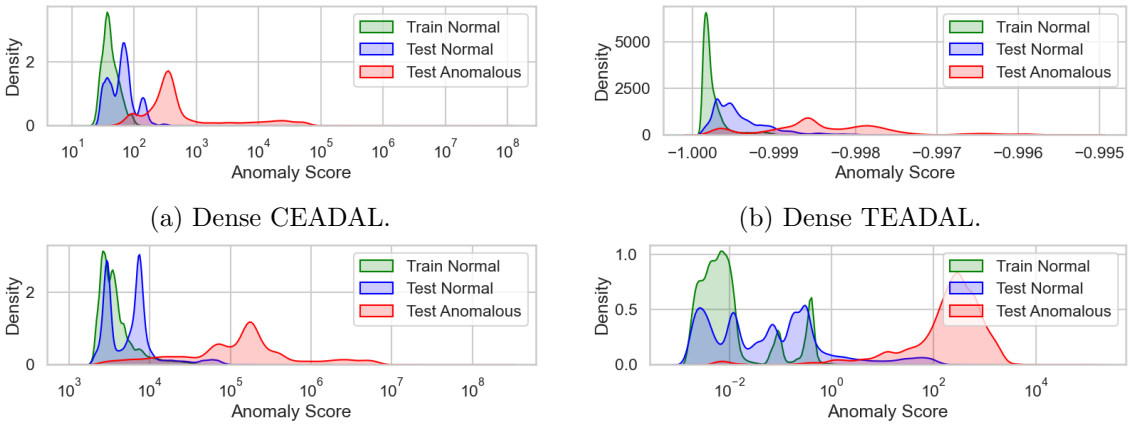
Table 6.1: Peak F1-scores achieved by the best-performing encoding and classification methods for each event type within a trace (averaged across test traces), with the top-three F1-scores for each event type shown in **bold**.

Improvements over Unsupervised Methods

Figure 6.4 shows that the three best-performing methods of this weakly-supervised study were Dense CEADAL, Conv DSAD, and Conv CEADAL, thus including both our CEADAL variants (**C1**). These three methods significantly improved on the unsupervised methods in terms of *median* performance (with 0.80, 0.76 and 0.75 median peak F1-scores, respectively, over 0.65 for TranAD, the best unsupervised baseline, and 0.68 for Dense DIVAD-GM, the best unsupervised domain generalization method), which shows that leveraging the few labeled anomalies was useful to achieve higher performances with less hyperparameter tuning (**C2**). The top-three methods also obtained higher *maximum* peak F1-scores than unsupervised methods (0.83, 0.81 and 0.80, respectively, over 0.66 for TranAD and 0.79 for Dense DIVAD-GM), showing that the few anomalies leveraged were also useful to reach higher performances with the right hyperparameter values (**C3**).

Implicit Context Generalization

An important factor that contributed to the superior performance of the CEADAL and Deep SAD variants was their higher robustness to the *normal behavior shift* challenge of the Exathlon benchmark, induced by their objective of learning similar latent representations for all the normal samples (**C4**). We highlight this implicit context generalization aspect in Figures 6.5a, 6.5b, 6.5c and 6.5d, showing Kernel Density Estimate (KDE) plots of the anomaly scores assigned to training normal, test normal and test anomalous records by Dense CEADAL, Dense TEADAL, Conv CEADAL and Conv DSAD, respectively. From these figures, we can see that the CEADAL and Deep SAD variants were typically more robust to normal behavior shift than the best TEADAL method, for which a similar latent representation of normal samples was not enforced. Specifically, we can see that CEADAL and Deep SAD variants assigned more similar anomaly scores to the *training* and *test* normal records (i.e., better aligned **green** and **blue** KDEs) than the best TEADAL method. This anomaly score alignment however remains imperfect, and weaker than the one achieved by the *explicit* domain generalization of Dense DIVAD-GM, shown in Figure 5.5 (**C5**). Another observation we can make from Figure 6.5b is that the richer representation of normal data learned by TEADAL, although less robust to normal behavior shift, resulted in a better modeling of the training data than CEADAL and Deep SAD (narrower **green** KDE), as we could expect from Section 6.2.4 (**C6**).



(a) Dense CEADAL.

(b) Dense TEADAL.

(c) Conv CEADAL.

(d) Conv DSAD.

Figure 6.5: Kernel Density Estimate (KDE) plots of the anomaly scores assigned by Dense CEADAL, Dense TEADAL, Conv CEADAL and Conv DSAD to training normal, test normal and test anomalous records.

Improved Anomaly Coverage

Considering the maximum peak F1-scores achieved by the methods in Table 6.1, we can see that **T1** (Bursty Input), **T2** (Bursty Input Until Crash), **T3** (Stalled Input) and **T5** (Driver Failure) were the **easiest types** of events to detect within test traces across the best encoding methods (**C7**). We can also see that our CEADAL variants achieved a good general separation of the different event types within the test traces, both with five out of six F1-scores in the top-three for their event type (**C8**).

Importantly, Table 6.1 further shows that the three best-performing methods (including our CEADAL variants) significantly improved on the maximum peak F1-scores of unsupervised methods for **T3** and **T4** (CPU Contention) events within test traces, hence partly addressing the challenge of *identifying the impacts of these harder events on our high-dimensional feature set* (**C9**). Specifically, Dense CEADAL, Conv DSAD and Conv CEADAL achieved peak F1-scores of 0.89, 0.91 and 0.89 for T3 events, respectively, over 0.67 and 0.80 for the best-performing point and sequence modeling unsupervised baselines (in Table 4.1), and 0.87 for the best-performing domain generalization method (in Table 5.1). Although sequence modeling methods still outperformed the point modeling ones for T3 events here, the significant improvement of Dense CEADAL over the point modeling unsupervised baselines shows the few labeled anomalies could facilitate point detection, being useful to identify the contextual and point anomalies induced by T3 events (**C10**). The largest overall improvement was however achieved for the (more subtle) T4 events (**C11**), with peak F1-scores of 0.85, 0.86 and 0.81 obtained for Dense CEADAL, Conv DSAD and Conv CEADAL, respectively, over 0.77 for Rec AE in Table 4.1, the best T4 peak F1-score across the unsupervised methods.

Again from Table 6.1, we can see that **T6** (Executor Failure) events however remained the **hardest type** to detect by far across the encoding and classification methods (**C12**). Specifically, the maximum peak F1-scores of 0.56, 0.61 and 0.57 obtained for Dense CEADAL, Conv DSAD and Conv CEADAL, respectively, did not improve on the best unsupervised T6 peak F1-score of 0.68, achieved by Rec DIVAD-GM in Table 5.1. This shortcoming could be explained by the relatively low representation of T6 anomalies in these methods' training and validation data (0.07% and 1.62% of samples, respectively, for $L = 1$, and 0.08% and 1.62% of samples, respectively, for $L = 20$), compared to the variety and detection difficulty of such events described in Section 4.3. From Tables 4.1, 5.1

and 6.1, we can see that T6 events tended to be better detected by sequence modeling methods, which indicates that T6 detection could be improved through a refined modeling only, besides altering our feature engineering strategy.

Finally, Table 6.1 shows that Conv TEADAL performed much worse than its fully-connected counterpart in terms of T1 and T3 event detection within traces, with 0.17 and 0.15 peak F1-scores compared to 0.96 and 0.59, respectively. This indicates that Conv TEADAL could require more hyperparameter tuning to reach a similar performance to Dense TEADAL in practice.

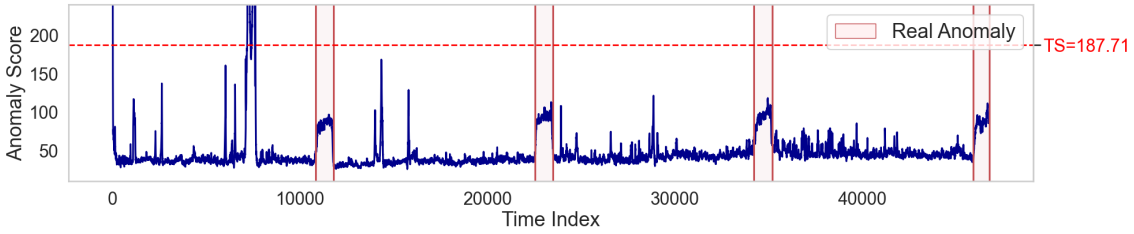
Improved Noise Integration

Figures 6.6a, 6.6b and 6.6c show the time plots of the anomaly scores assigned by Dense CEADAL, Conv DSAD and Conv CEADAL, respectively, in the (Stalled Input) trace `6_3_200000_76`, whose first half contains a particularly high level of noise. From these figures, and Figure 4.15, we can see that these methods were far more able to integrate heavy and sustained noisy patterns into their normal behavior than TranAD, the "most robust" unsupervised baseline from Chapter 4. This indicates that leveraging the few anomaly labels as prior knowledge for the anomaly types to detect was indeed useful in *differentiating noisy, minority patterns from our relevant anomalies in test data*, hence addressing our corresponding challenge (**C13**). We can also see this resilience to noise was even higher for sequence modeling methods, relying on larger contexts of length $L = 20$, in which small irrelevant deviations get diminished, to derive their anomaly scores (**C14**). Like for TranAD, most of the anomaly scores assigned by the three methods in this trace were however below their peak F1-score threshold, which resulted in them missing all the corresponding T3 events, and again highlights the suboptimality of their implicit context generalization to address our normal behavior shift challenge.

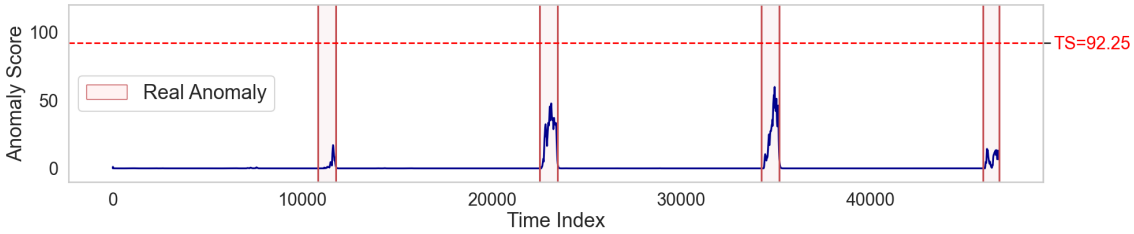
Limitations of Classification Methods

Another observation we can make from Figure 6.4 and Table 6.1 is that classification methods performed much worse than our best encoding methods, but also worse than the best *unsupervised* baselines from Chapter 4 (**C15**). Only Dense DIVA could indeed slightly outperform TranAD in maximum peak F1-score, with 0.67 over 0.65. Like all other classification methods, it however performed worse in median peak F1-score, with 0.63 compared to 0.65 for TranAD.

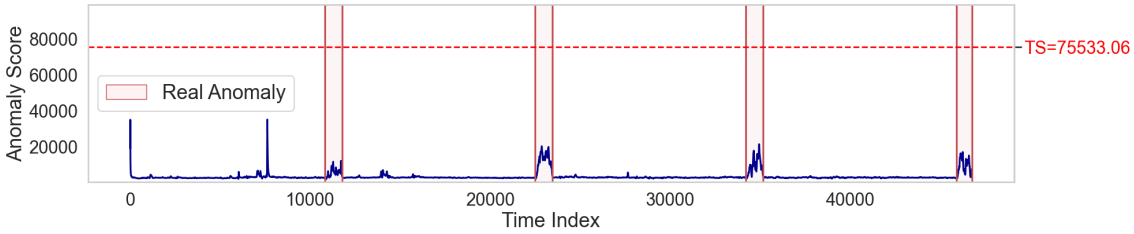
A first factor explaining this poorer performance can be identified from Table 6.1, showing that the three classification methods could not detect most of **T5** (Driver Failure) events within the test traces (**C16**), with peak F1-scores of 0.18, 0.11 and 0.31 obtained for XGBoost, Dense DIVA and Conv DIVA, respectively. Figures 6.7 and 6.8b, showing the time plots of the anomaly scores assigned by XGBoost in trace `6_5_1000000_93` and Dense DIVA in trace `3_5_1000000_89`, respectively, further highlight the inability of these methods to detect T5 anomaly signals. This limitation can most likely be explained by the very low representation of T5 anomalies in the training and validation data, constituting only 0.01% and 0.01% of samples, respectively, for $L = 1$, and 0.02% and 0.01% of samples, respectively, for $L = 20$. Because encoding methods primarily focus on modeling the *normal behavior* of the data, and deem test samples anomalous if they *deviate* from this normal behavior, they typically require very few, if any, training examples of T5 anomalies to accurately detect them from the significant deviations they induce in test samples. In contrast, classification methods focus on modeling the *normal and anomalous classes* individually, deeming test samples anomalous if they belong to the specific "anomaly class" learned from training. As such, we can expect these methods to require more training examples of T5 anomalies to accurately detect them in test data.



(a) Anomaly scores of Dense CEADAL.



(b) Anomaly scores of Conv DSAD.



(c) Anomaly scores of Conv CEADAL.

Figure 6.6: Time plots of the anomaly scores assigned by Dense CEADAL, Conv DSAD and Conv CEADAL in trace 6_3_200000_76, highlighting their peak F1-score thresholds and the ground-truth anomaly ranges.

For XGBoost, this limitation to detect T5 events was further combined with a vulnerability to the *normal behavior shift* of the Exathlon benchmark, showing that simply leveraging a few anomaly labels was not helpful in generalizing to different trace contexts (**C17**). This lack of generalization is illustrated in Figure 6.9, showing the ridgeline plot of XGBoost's anomaly scores for the records of each type (i.e., in "normal" and T1 to T6 ranges) in each test trace, with its peak F1-score threshold highlighted in red. From this figure, we can see that, although XGBoost was able to separate most of the anomalous records from normal data within the test traces, the *misalignment* of anomaly scores assigned to normal records across different traces prevented it from achieving optimal performance with a single, global threshold. Clear examples from Figure 6.9 include the test traces 6_3_200000_76 and 10_5_1000000_85, for which the anomaly scores assigned by XGBoost were nearly all below its optimal global threshold, leading it to miss most of the corresponding anomalies despite achieving a reasonable separation within each trace.

For DIVA, another factor that seemed to hinder the performance was the output activations of its classification head NN_{ω_y} used to derive the anomaly scores (**C18**). When using sigmoid or softmax activation functions, classification methods indeed tend to *bias* their model toward a specific anomaly score threshold as part of their training procedure. As such, we can expect them to benefit far less from our "peak F1-score" evaluation strategy, since tuning their threshold based on the test performance will be far less effective. This is illustrated for Dense DIVA in Figure 6.6, showing that its peak F1-score threshold was set

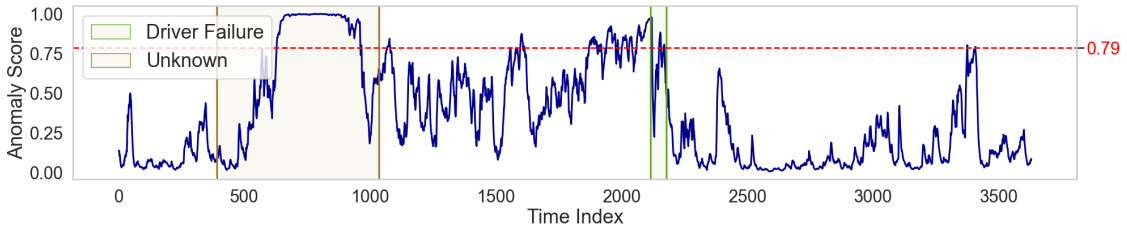


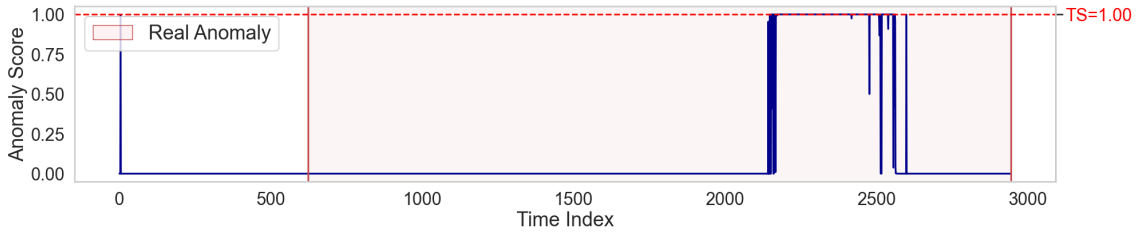
Figure 6.7: Time plot of the anomaly scores assigned by XGBoost in trace 6_5_1000000_93, highlighting its peak F1-score threshold and the ground-truth anomaly ranges.

to (the already "pre-thresholded" value of) 1.0. This "dichotomous" behavior, also present to a lesser extent for XGBoost, as shown in Figure 6.9, had the effect of increasing both the *false negative* and *false positive* rates of Dense DIVA. Figures 6.8a and 6.8c, showing the time plots of the anomaly scores assigned by Dense DIVA in traces 1_2_100000_68 and 6_5_1000000_93, respectively, present illustrative examples of this phenomenon. In Figure 6.8a, we can see that (possibly less-significant) abnormal patterns in the T2 (Bursty Input Until Crash) range of trace 1_2_100000_68 tended to get assigned very low anomaly scores, producing a significant amount of false negative predictions. Figure 6.8c highlights the opposite behavior, with (possibly slightly-deviating) normal patterns outside the real anomaly ranges of trace 6_5_1000000_93 getting assigned very high anomaly scores, which produced a lot of false positives. For traditional anomaly detection methods, records from these two cases were typically assigned more "nuanced" anomaly scores, which allowed our evaluation strategy to find threshold values that mitigated these false negative and false positive issues.

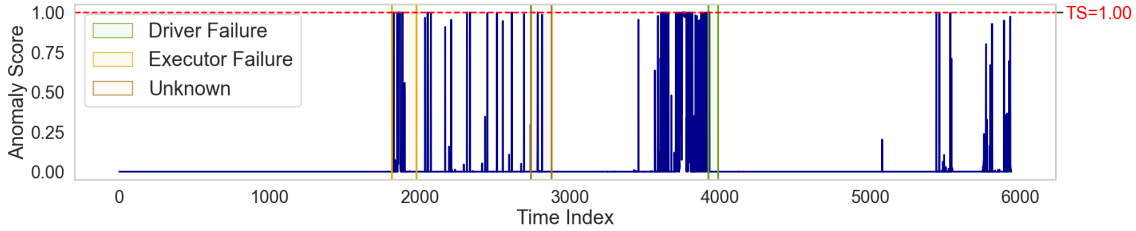
6.4 Summary and Conclusions

In this chapter, we started by formalizing a revised problem statement for anomaly detection under a **weakly-supervised** setting, where a few anomaly labels are available to train the anomaly detection methods. We then proposed a method called **CEADAL**, relying on contrastive learning to construct a latent mapping where normal data samples are grouped within a tight region, away from the anomalies. We also presented **TEADAL**, an alternative of CEADAL based on the triplet loss, inducing richer, less-constrained representations of normal data samples in latent space. We then applied our CEADAL and TEADAL methods against the Exathlon benchmark under this new weakly-supervised setting, comparing them to other encoding and classification methods. After presenting the methods compared and their grid of hyperparameter values, we presented our results and analyses from this study, which led to **18 conclusions**, summarized and grouped by similarity below:

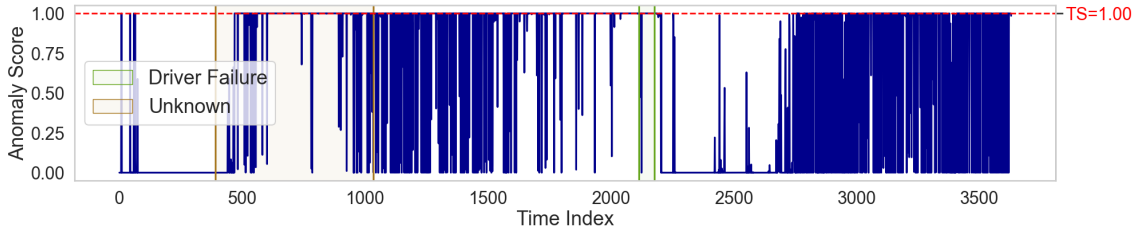
- **C1, C2, C3.** The three best-performing methods were Dense CEADAL, Conv DSAD, and Conv CEADAL, thus including both our CEADAL variants. Leveraging a few labeled anomalies was both useful in achieving (i) a higher performance with less hyperparameter tuning, and (ii) a higher performance with properly tuned hyperparameter values.
- **C4, C5.** An important factor in the superiority of CEADAL and Deep SAD variants over the other methods was their higher ability to handle the *normal behavior shift* challenge of the Exathlon benchmark. This ability stemmed from the implicit context generalization induced by these methods, itself resulting from the strong proximity



(a) Anomaly scores in trace 1_2_100000_68.



(b) Anomaly scores in trace 3_5_1000000_89.



(c) Anomaly scores in trace 6_5_1000000_93.

Figure 6.8: Time plots of the anomaly scores assigned by Dense DIVA in traces 1_2_100000_68, 3_5_1000000_89 and 6_5_1000000_93, highlighting its peak F1-score threshold and the ground-truth anomaly ranges.

constraints they enforced for the normal samples in latent space. Because of its implicit nature, *this generalization however remained suboptimal*, and weaker to the one reached by our Dense DIVAD-GM method from Chapter 5.

- **C6.** Despite being less robust to normal behavior shift, the richer representation learned by TEADAL for the normal samples resulted in a better modeling of the training data than the other methods.
- **C7, C8, C9, C10, C11.** Within test traces, the easiest types of events to detect across the best encoding methods were T1 (Bursty Input), T2 (Bursty Input Until Crash), T3 (Stalled Input) and T5 (Driver Failure), with our CEADAL variants achieving a good general separation of the different event types. The best-performing methods (including CEADAL) significantly improved on the maximum peak F1-scores of unsupervised methods for T3 and T4 (CPU Contention) events, hence partly addressing the challenge of *identifying the impacts of these harder events on our high-dimensional feature set*. The largest overall improvement was achieved for the (more subtle) T4 events, but the few labeled anomalies were also helpful in better detecting T3 events with point modeling methods.
- **C12.** Within test traces, *the hardest type of events to detect across the methods remained T6 (Executor Failure)*. This could be explained by the relatively low representation of T6 anomalies in the training and validation data, compared to the variety and detection difficulty of these events.

- **C13, C14.** The best encoding methods of this weakly-supervised study were far more able to *differentiate normal but noisy, minority patterns from relevant anomalies* than unsupervised methods, showing that the few labeled anomalies were indeed useful to contrast relevant anomalies from other minority patterns. This robustness to noise was even higher for sequence modeling methods, which could rely on larger contexts to derive their anomaly scores.
- **C15, C16, C17, C18.** The classification methods performed much worse than our best encoding methods, but also worse than the best unsupervised baselines. The first factor of this poorer performance was the inability of classification methods to detect most of T5 (Driver Failure) events, most likely due to their lack of representation in the training and validation data. For XGBoost, this limitation was further combined with a vulnerability to the normal behavior shift challenge of our benchmark. For DIVA, and to some extent all classification methods, it was further combined with the negative impact of the "pre-thresholding" performed for its predictions during training, disadvantaged by our evaluation strategy, which rely on finding the optimal threshold for the anomaly scores in test data.

In particular, this study showed that our weakly-supervised setting and proposed CEADAL method could partly address all three of the limitations **L1-3** identified for the unsupervised baselines in Chapter 4, providing a solution that (i) *is more robust to the normal behavior shift from training to test data*, (ii) *better identifies anomaly signals of harder event types in a high-dimensional feature set*, and (iii) *better distinguishes normal but noisy, minority patterns from our relevant event types in test data*.

This study also revealed some **improvement directions** for our proposed methods which could be pursued in future work, as further discussed in the next chapter.

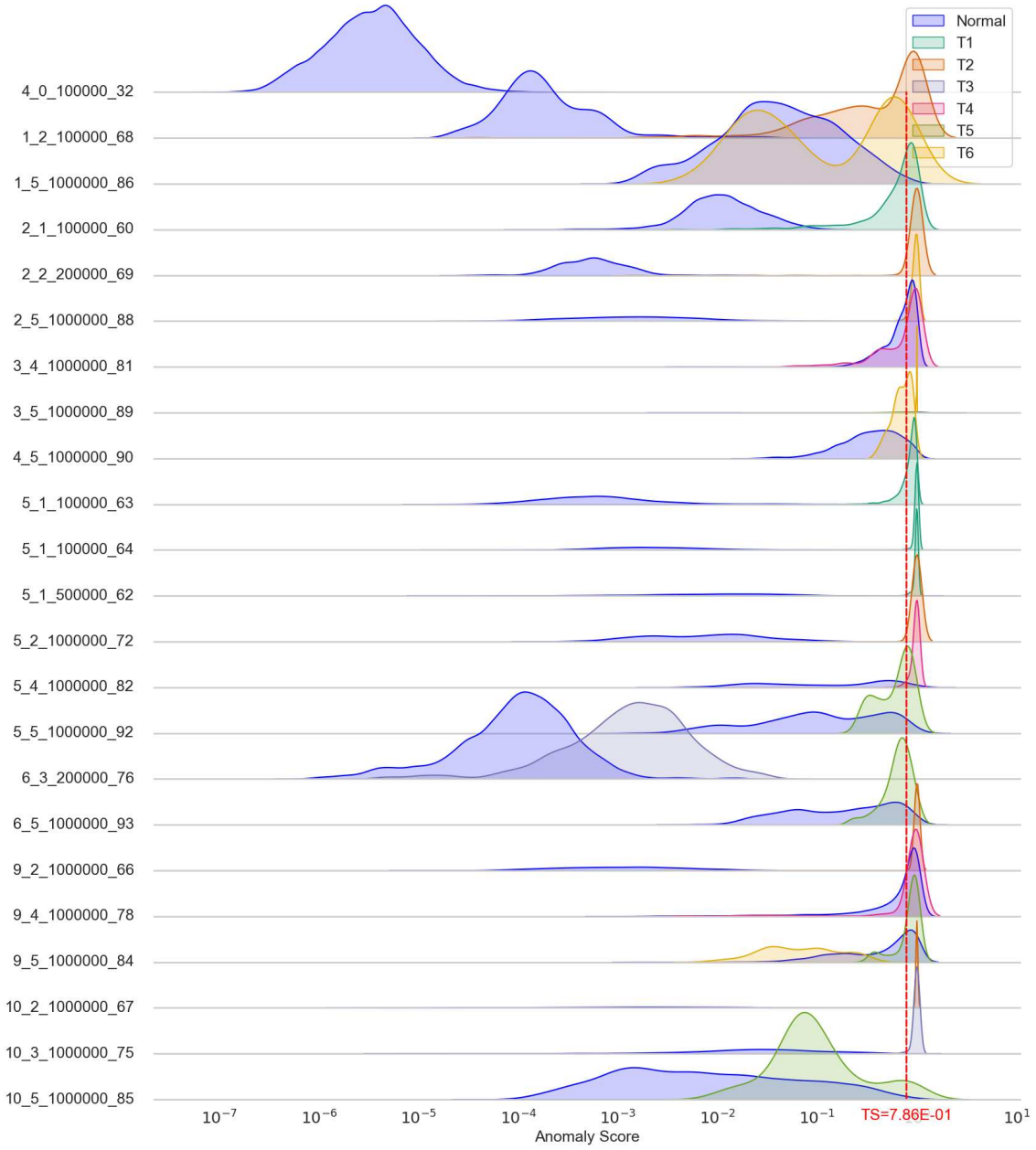


Figure 6.9: Ridgeline plot of XGBoost's anomaly scores for the records of each type (i.e., in "normal" and T1 to T6 ranges) in each test trace, with its peak F1-score threshold highlighted in red.

7 Conclusions and Perspectives

This chapter summarizes the main contributions and conclusions of this thesis, in Section 7.1, and lays out some perspectives for future work, in Section 7.2.

7.1 Conclusions

The first objective of this thesis was to address the lack of tools openly available to develop and benchmark explainable anomaly detection methods in high-dimensional time series, with a focus on AIOps scenarios. We could achieve this objective through the following **contribution**:

CTB1 *New benchmarking tools for explainable anomaly detection in high-dimensional time series*, including the **Exathlon** benchmark[1] and the design of LEADS Viewer[2]. The Exathlon benchmark first includes a Spark Streaming dataset, consisting of data traces of significantly higher dimensionality than traditional time series datasets (around 2.3M raw data records of 2,283 columns, brought to 2.2M with 237 features in our experimental setup), collected with AIOps considerations in mind. Through its disturbed traces, this dataset provides labeled anomaly intervals for six different types of events T1-6, enabling the detailed analysis of anomaly detection (AD) and explanation discovery (ED) methods. Exathlon also includes a flexible evaluation methodology for AD and ED. Its AD functionality implements metrics for both range-based and point-based evaluation, with the option of using different AD levels to reflect increasingly challenging requirements. Its ED functionality provides new metrics to evaluate anomaly explanations both "locally" (for a given explained instance) and "globally" (across multiple explained instances), without the need for ground-truth explanatory features. Finally, Exathlon provides a full pipeline for explainable anomaly detection, designed to be modular within and beyond its Spark Streaming use case, allowing different datasets and methods to be added and combined. Either jointly or independently from Exathlon, LEADS Viewer can be used as an effective way to qualitatively assess explainable anomaly detection methods, providing an easy-to-use Graphical User Interface (GUI) to visualize their outputs on high-dimensional time series data.

The remainder of this thesis focused on the anomaly detection functionality of the Exathlon benchmark, starting with the objective of assessing the performance of representative unsupervised methods against its AIOps use case. This objective was mainly achieved through the following **contribution**:

CTB2 *An in-depth benchmarking analysis of representative unsupervised anomaly detection methods*. This study started with a thorough analysis of the Spark Streaming dataset, deriving a detailed experimental setup showed to reflect key AIOps challenges. It then derived 15 conclusions for the methods compared, covering their characteristics and trade-offs, ability to detect different types of events, and limitations to address the benchmark challenges. Distribution methods were for instance more vulnerable to *normal behavior shift* from training to test data than reconstruction methods due to their explicit modeling

[1] Publicly available at `https://github.com/exathlonbenchmark/exathlon`.
[2] Publicly available at `https://github.com/exathlonbenchmark/leads-viewer`.

of the normal training distribution, which resulted in lower performance (with a peak F1-score of 0.54 for the best distribution method vs. 0.66 for the best reconstruction method). Another conclusion was that the larger contexts used by sequence modeling methods to derive their anomaly scores usually made them more robust to *"noisy", minority patterns* than point modeling methods. Across the methods compared, T1 (Bursty Input), T2 (Bursty Input Until Crash) and T5 (Driver Failure) events were typically easier to detect than T3 (Stalled Input), T4 (CPU Contention) and T6 (Executor Failure) events, with T6 events being the most challenging type. Overall, this study revealed the unsupervised methods were limited to solve our benchmark setup, with a maximum peak F1-score of only 0.66 achieved by TranAD [14]. In particular, the methods compared displayed three main **limitations**, listed below in decreasing order of impact on the AD performance:

- **L1.** *A vulnerability to normal behavior shift from training to test data*, with different *contexts* of *normal* operation for the Spark Streaming applications being considered differently abnormal by the methods.
- **L2.** *A production of false negatives for the hardest anomalies given our large number of features*.
- **L3.** *A production of false positives for normal but "noisy", minority patterns in test data*.

The third objective of this thesis was therefore to explicitly address **L1** in this unsupervised setting. We tackled it through the approach of *domain generalization*, considering different normal behaviors as *heterogeneous domains*, and associating *normal behavior shift* to the concept of *domain shift*. This approach allowed us to effectively achieve our objective, through the following **contribution**:

CTB3 *A new explicit domain generalization method for unsupervised anomaly detection*. We started by formally characterizing the problem of unsupervised anomaly detection under domain shift. We then proposed Domain-Invariant VAE for Anomaly Detection, or **DIVAD**, relying on feature disentanglement to decompose the observed variable into *domain-specific* and *domain-invariant* encodings, and defining anomalies as samples that deviate from the training distribution of domain-invariant encodings only. We designed and run different DIVAD variants against our benchmark and setup, and showed that the normal behavior learned by the best-performing ones generalized much better to the test data than the unsupervised baselines, thus addressing their limitation **L1** and significantly improving their AD performance (with a maximum peak F1-score of 0.79). Through its 14 conclusions, our study also confirmed that using a learned Gaussian Mixture prior (DIVAD-GM) over a fixed Gaussian prior (DIVAD-G) for the domain-invariant encoding variable was always beneficial, although DIVAD-G could still outperform the unsupervised baselines when deriving its anomaly scores from an aggregated posterior estimate instead of the prior (with a maximum peak F1-score of 0.76 obtained with the aggregated posterior estimate vs. 0.64 obtained with the prior). Another conclusion was that using point modeling DIVAD variants was here sufficient to outperform the unsupervised baselines, due to the predominantly contextual nature of the anomalies in our setup. We finally applied DIVAD to the Application Server Dataset (ASD) [13], and showed it could outperform the best unsupervised baseline TranAD in maximum peak F1-score for 92% of the test cases (with over 10% improvements for 67% of them), hence highlighting its broader applicability beyond our Spark Streaming dataset.

Despite their success in addressing **L1**, none of our DIVAD variants could effectively tackle the remaining limitations **L2** and **L3** of the unsupervised baselines, showing no improvement in the detection of more complex T4 and T6 events, nor being particularly more resilient to "noisy", minority patterns encountered in test data. The fourth objective

of this thesis was therefore to jointly address the three limitations **L1-3** in a *weakly-supervised* extension of our setting, where a few labeled anomalies are available to train the detection methods. We could take a significant step toward this objective, through the following **contribution**:

CTB4 *New contrastive methods in a weakly-supervised setting*. We proposed Contrastive Encoder for Anomaly Detection with a Few Anomaly Labels (**CEADAL**), a method that (i) extracts normal-normal and normal-anomaly *pairs* from the training samples, (ii) relies on the *contrastive loss* to construct a latent mapping where normal data samples are grouped within a tight region, away from the anomalies, and (iii) defines anomalies as samples that deviate from the training distribution of normal latent samples. To study the impact of the contrastive loss used by CEADAL, we also proposed an alternative method based on the *triplet loss* called TEADAL, inducing richer, less-constrained representations of normal data samples in latent space. We then applied CEADAL and TEADAL against the Exathlon benchmark, under the experimental setup of our unsupervised study augmented with training anomalies, and compared them with other encoding and classification methods. Our study derived 18 conclusions, and showed that CEADAL was very effective in addressing the three limitations **L1-3**, achieving the highest maximum and median peak F1-scores across all the methods compared (0.83 and 0.80, respectively). In particular, our study confirmed that the strong proximity constraints enforced by CEADAL for normal samples in latent space tended to induce an "implicit context generalization" that benefited addressing **L1**. This improved generalization in turn explained its higher performance achieved compared to less-constrained alternatives like TEADAL, which obtained a maximum peak F1-score of only 0.64 despite a better modeling of the training data. The few labeled anomalies leveraged by CEADAL were also helpful in better detecting T4 (CPU Contention) events, as well as differentiating even the most sustained noisy patterns from our relevant anomalies in test data, thereby tackling **L2** and **L3**. Our study finally outlined several reasons for the poorer performance of the classification methods, including (i) their inability to detect under-represented T5 (Driver Failure) events, (ii) their vulnerability to *normal behavior shift* (for XGBoost [129]), and (iii) the "pre-thresholding" they performed during training, which was disadvantaged by our evaluation strategy.

Our CEADAL method significantly improved the maximum and median AD performance of the unsupervised baselines, making notable progress in addressing their limitations **L1-3**. Because of its implicit nature, *the context generalization it performed*, and thus its handling of **L1**, however *remained suboptimal*, and less effective than the one we could reach using our DIVAD framework. Besides, *none of the methods could improve the detection of T6 (Executor Failure) events*, and thus fully overcome **L2**. These remaining challenges highlight important research directions for future work on the modeling side, which will be further discussed in Section 7.2.

7.2 Perspectives

The work of this thesis could be extended and pursued in the following directions:

Explainable anomaly detection benchmarking. Being extensible and modular, the Exathlon benchmark could be expanded with additional, more recent evaluation metrics in future work. For anomaly detection, these could include Volume Under the Surface (VUS) [114], a recent metric offering benefits such as a greater robustness to lag and noise, as well as a clear separation between "accurate" and "inaccurate" methods. In our experimental studies, we chose to consider the simplest metrics that aligned with our requirements and desired properties for the compared methods. Incorporating VUS in

Exathlon could however be useful to extend these studies, as well as in other scenarios. On the explanation front, our benchmark could be expanded to support more Co-12 properties [39] and explanation formats. *Target Sensitivity* could for instance be better captured by requiring *high* Discordance(·) scores across explanations of *different* event types, in addition or replacement to *low* scores across explanations of a *same* type. Multiple perturbation mechanisms could also be added and compared for our Instability(·) metric, and consider the impact the perturbed instances have on corresponding model predictions. We finally hope the release of Exathlon's Spark Streaming dataset will be part of a growing effort to openly release challenging, real-world data traces for explainable AD in AIOps. As our studies showed, transparency about the precise settings and environment used to record such traces is instrumental in characterizing their *contexts* of normal operation, and thus a significant part of the problem. Providing a wide set of anomalous events is also valuable, as it enables studying the behavior of different explainable AD methods at a finer level. To further improve the realism and quality of explanation discovery, it is also important for these events to induce slight, *progressive* effects on the recorded entities, to enable identifying event type-specific precursory signals of abnormality before Key Performance Indicators (KPIs) get significantly impacted.

Anomaly detection and context generalization. Despite its success in addressing the main limitation **L1** of our baselines, the unsupervised nature of our DIVAD framework makes it vulnerable to removing relevant anomaly signals from the samples in domain-invariant space. This was for instance the case for Rec DIVAD-G in our study, which significantly hindered its coverage of T1 (Bursty Input) events. The "anomaly-aware" mapping constructed through our weakly-supervised setting and CEADAL method could alleviate this issue, but with a *weaker, implicit generalization* and addressing of **L1**. Efforts have been made during this thesis to design a *weakly-supervised* version of DIVAD, combining its explicitly tackling of *normal behavior shift* with the robustness to removing anomaly signals brought by the few labeled anomalies. Although the performance obtained was not yet satisfactory, we believe this approach could still improve the current results. Further efforts could also be directed at making the DIVAD framework and its weakly-supervised version more stable to reduce the amount of hyperparameter tuning needed. Additionally, more complex architectures could be considered in the future to better detect *more complex event types like T6 (Executor Failure)*, since we could see that more advanced sequence modeling methods were beneficial in this regard. During this thesis, we also experimented with a time-frequency architecture for DIVAD, considering time and frequency "paths" within its encoders and decoder. The frequency paths operated on the Discrete Fourier Transform (DFT) of the input samples, which can typically be more domain-invariant than time-based features [113]. Although experiments using this architecture across multiple window lengths and downsampling factors could not improve the performance so far, it could still be further explored in the future for this or other scenarios. Finally, another interesting direction to address **L1** could be through the scope of *fast domain adaptation*, where a base model can be adjusted to the first few data samples of a given test trace (i.e., target domain). Although this approach could be more costly than domain generalization due to this "retraining" step, the cost would still be lower than training a model per trace. It could also be further reduced, for instance relying on AD to identify whether the domain shift of a test trace is significant enough to require an adjustment. Such cost analyses for model training and inference would also constitute important aspects to consider as extensions of our studies, which focused more on the effectiveness of the AD methods in addressing our AIOps challenges.

Explainability. The significant challenges encountered for the compared AD methods with respect to our benchmark centered the modeling work of this thesis around the

anomaly detection component of our AIOps use case. Its *explainability* component yet remains essential to indicate the reasons behind anomalies, so that operators like site reliability engineers (SREs) can better understand the issues of the monitored entities and deploy corrective actions. During this thesis, some work was for instance done to help improve the EXstream algorithm [90], proposing to explain a given anomalous range of records with respect to a reference range, where the anomalous range was typically flagged by a detection method. An advantage of EXstream is that it can operate fully independently of the explained model. As such, it can optionally rely on a different feature set, considering additional metrics that would be harmful to model globally but useful to consider locally in search for a root cause. For our Spark Streaming dataset, this could correspond to the OS metrics recorded using `nmon`. These metrics were ignored when constituting the feature set of our experimental setup, since they relate to the entire cluster and are thereby influenced by all the concurrent jobs running on it. However, they could still be useful to further diagnose what is occurring during an anomalous event on the cluster, outside the application, and potentially facilitate going back to a root cause. Explainability could also be added to our existing methods by updating and extending the experimental study of data and model explainers we conducted in our Exathlon paper [17], considering either the current or some extended metrics for ED, as discussed earlier. Besides data and model-agnostic explainers, methods relying on model internals could also be included, for instance considering attention-based architectures for our DIVAD models, and working on making attention weights reflect desirable properties of explanations [131].

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Y. Dang, Q. Lin, and P. Huang, "Aiops: Real-world challenges and research innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 4–5, 2019.

[3] M. S. (IDC), "Worldwide spending on public cloud services is forecast to double between 2024 and 2028, according to new idc spending guide." `https://www.idc.com/getdoc.jsp?containerId=prUS52460024`, 2024. Accessed: 2024-08-01.

[4] G. Kim, P. Debois, J. Willis, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[5] Gartner, "Market guide for aiops platforms." `https://www.gartner.com/en/documents/3772124`, 2018. Accessed: 2023-10-10.

[6] Z. Zhong, Q. Fan, J. Zhang, M. Ma, S. Zhang, Y. Sun, Q. Lin, Y. Zhang, and D. Pei, "A survey of time series anomaly detection methods in the aiops domain," 2023.

[7] M. Intelligence, "Aiops platforms market size - industry report on share, growth trends & forecasts analysis (2024 - 2029)." `https://www.mordorintelligence.com/industry-reports/aiops-market`, 2024. Accessed: 2024-08-01.

[8] V. Chandola *et al.*, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[9] Z. Zhong, Q. Fan, J. Zhang, M. Ma, S. Zhang, Y. Sun, Q. Lin, Y. Zhang, and D. Pei, "A survey of time series anomaly detection methods in the aiops domain," 2023.

[10] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 38–44, 2015.

[11] K.-H. Lai, D. Zha, J. Xu, Y. Zhao, G. Wang, and X. Hu, "Revisiting time series outlier detection: Definitions and benchmarks," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (J. Vanschoren and S. Yeung, eds.), vol. 1, Curran, 2021.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[13] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei, "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, (New York, NY, USA), p. 3220–3230, Association for Computing Machinery, 2021.

[14] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: Deep transformer networks for anomaly detection in multivariate time series data," *Proc. VLDB Endow.*, vol. 15, p. 1201–1214, feb 2022.

[15] M. Jiang, C. Hou, A. Zheng, X. Hu, S. Han, H. Huang, X. He, P. S. Yu, and Y. Zhao, "Weakly supervised anomaly detection: A survey," 2023.

[16] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National Science Review*, vol. 5, pp. 44–53, 08 2017.

[17] V. Jacob *et al.*, "Exathlon: A benchmark for explainable anomaly detection over time series," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2613–2626, 2021.

[18] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, jul 2009.

[19] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: A comprehensive evaluation," *Proc. VLDB Endow.*, vol. 15, p. 1779–1797, may 2022.

[20] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, p. 4396–4415, Apr. 2023.

[21] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and P. S. Yu, "Generalizing to unseen domains: A survey on domain generalization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 8, pp. 8052–8072, 2023.

[22] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 1735–1742, 2006.

[23] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[24] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, pp. 85–126, 10 2004.

[25] V. Chandola, V. Mithal, and V. Kumar, "Comparative evaluation of anomaly detection techniques for sequence data," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 743–748, 2008.

[26] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017. Online Real-Time Learning Strategies for Data Streams.

[27] N. Singh and C. Olinsky, "Demystifying numenta anomaly benchmark," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1570–1577, 2017.

[28] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*, 2018.

[29] S. Moritz, F. Rehbach, S. Chandrasekaran, M. Rebolledo, and T. Bartz-Beielstein, "Gecco industrial challenge 2018 dataset: A water quality dataset for the 'internet of things: Online anomaly detection for drinking water quality' competition," in *Genetic and Evolutionary Computation Conference*, feb 2018.

[30] J. Paparrizos, Y. Kang, P. Boniol, R. S. Tsay, T. Palpanas, and M. J. Franklin, "Tsb-uad: An end-to-end benchmark suite for univariate time-series anomaly detection," *Proc. VLDB Endow.*, vol. 15, p. 1697–1711, apr 2022.

[31] P. Wenig, S. Schmidl, and T. Papenbrock, "Timeeval: A benchmarking toolkit for time series anomaly detection algorithms," *Proc. VLDB Endow.*, vol. 15, p. 3678–3681, aug 2022.

[32] S. Alnegheimish, D. Liu, C. Sala, L. Berti-Equille, and K. Veeramachaneni, "Sintel: A machine learning framework to extract insights from signals," in *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, (New York, NY, USA), p. 1855–1865, Association for Computing Machinery, 2022.

[33] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, (New York, NY, USA), p. 2828–2837, Association for Computing Machinery, 2019.

[34] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, (New York, NY, USA), p. 387–395, Association for Computing Machinery, 2018.

[35] C. M. Ahmed, V. R. Palleti, and A. P. Mathur, "Wadi: A water distribution testbed for research in the design of secure cyber physical systems," in *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, CySWATER '17, (New York, NY, USA), p. 25–28, Association for Computing Machinery, 2017.

[36] A. P. Mathur and N. O. Tippenhauer, "Swat: a water treatment testbed for research and training on ics security," in *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, pp. 31–36, 2016.

[37] C. Molnar, *Interpretable Machine Learning*. Lulu.com, 2 ed., 2022.

[38] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv: Machine Learning*, 2017.

[39] M. Nauta, J. Trienes, S. Pathak, E. Nguyen, M. Peters, Y. Schmitt, J. Schlötterer, M. van Keulen, and C. Seifert, "From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable ai," *ACM Comput. Surv.*, vol. 55, jul 2023.

[40] P. Q. Le, M. Nauta, V. B. Nguyen, S. Pathak, J. Schlötterer, and C. Seifert, "Benchmarking explainable ai - a survey on available toolkits and open challenges," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23* (E. Elkind, ed.), pp. 6665–6673, International Joint Conferences on Artificial Intelligence Organization, 8 2023. Survey Track.

[41] N. Myrtakis, V. Christophides, and E. Simon, "A Comparative Evaluation of Anomaly Explanation Algorithms," in *24th International Conference on Extending Database Technology (EDBT'2021)*, (Nicosia, Cyprus), Mar. 2021.

[42] FICO, "Explainable machine learning challenge." https://community.fico.com/s/explainable-machine-learning-challenge, 2018. Accessed: 2023-11-27.

[43] K. N. Markelle Kelly, Rachel Longjohn, "The uci machine learning repository." https://archive.ics.uci.edu. Accessed: 2023-11-28.

[44] F. Keller, E. Muller, and K. Bohm, "Hics: High contrast subspaces for density-based outlier ranking," in *2012 IEEE 28th International Conference on Data Engineering*, pp. 1037–1048, 2012.

[45] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, "Explainability methods for graph convolutional neural networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10764–10773, 2019.

[46] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "Adbench: Anomaly detection benchmark," in *Advances in Neural Information Processing Systems* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds.), vol. 35, pp. 32142–32159, Curran Associates, Inc., 2022.

[47] Z. Z. Darban, G. I. Webb, S. Pan, C. C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," 2022.

[48] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2014.

[49] P. Malhotra, L. Vig, G. M. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *The European Symposium on Artificial Neural Networks*, 2015.

[50] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *AAAI Conference on Artificial Intelligence*, 2021.

[51] C. C. Aggarwal, *Linear Models for Outlier Detection*, pp. 65–110. Cham: Springer International Publishing, 2017.

[52] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier detection using replicator neural networks," in *Data Warehousing and Knowledge Discovery* (Y. Kambayashi, W. Winiwarter, and M. Arikawa, eds.), (Berlin, Heidelberg), pp. 170–180, Springer Berlin Heidelberg, 2002.

[53] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, (New York, NY, USA), p. 4–11, Association for Computing Machinery, 2014.

[54] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4393–4402, PMLR, 10–15 Jul 2018.

[55] Y. Yang, C. Zhang, T. Zhou, Q. Wen, and L. Sun, "Dcdetector: Dual attention contrastive representation learning for time series anomaly detection," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, (New York, NY, USA), p. 3033–3045, Association for Computing Machinery, 2023.

[56] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," in *Proceedings of International Conference on Data Mining*, 01 2003.

[57] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," in *Special Lecture on IE*, 2015.

[58] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008.

[59] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate time-series anomaly detection via graph attention network," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 841–850, 2020.

[60] L. Wong, D. Liu, L. Berti-Equille, S. Alnegheimish, and K. Veeramachaneni, "Aer: Auto-encoder with regression for time series anomaly detection," in *2022 IEEE International Conference on Big Data (Big Data)*, (Los Alamitos, CA, USA), pp. 1152–1161, IEEE Computer Society, dec 2022.

[61] C. C. Aggarwal, *An Introduction to Outlier Analysis*, pp. 1–34. Cham: Springer International Publishing, 2017.

[62] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," in *International Conference on Learning Representations*, 2020.

[63] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "Cade: Detecting and explaining concept drift samples for security applications," in *Proc. of USENIX Security*, 2021.

[64] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *J. Mach. Learn. Res.*, vol. 17, p. 2096–2030, jan 2016.

[65] T. Matsuura and T. Harada, "Domain generalization using a mixture of multiple latent domains," in *AAAI Conference on Artificial Intelligence*, 2019.

[66] K. Akuzawa, Y. Iwasawa, and Y. Matsuo, "Adversarial invariant feature learning with accuracy constraint for domain generalization," in *Machine Learning and Knowledge Discovery in Databases* (U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, eds.), (Cham), pp. 315–331, Springer International Publishing, 2020.

[67] R. Shao, X. Lan, J. Li, and P. C. Yuen, "Multi-adversarial discriminative deep domain generalization for face presentation attack detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[68] Y. Li, X. Tian, M. Gong, Y. Liu, T. Liu, K. Zhang, and D. Tao, "Deep domain generalization via conditional invariant adversarial networks," in *Computer Vision – ECCV 2018* (V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds.), (Cham), pp. 647–663, Springer International Publishing, 2018.

[69] N. Kodali, J. Hays, J. D. Abernethy, and Z. Kira, "On convergence and stability of gans," *arXiv: Artificial Intelligence*, 2018.

[70] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, "Stabilizing training of generative adversarial networks through regularization," in *Neural Information Processing Systems*, 2017.

[71] M. Ilse, J. M. Tomczak, C. Louizos, and M. Welling, "Diva: Domain invariant variational autoencoders," in *Proceedings of the Third Conference on Medical Imaging with Deep Learning*, vol. 121 of *Proceedings of Machine Learning Research*, pp. 322–348, PMLR, 06–08 Jul 2020.

[72] F. Qiao, L. Zhao, and X. Peng, "Learning to learn single domain generalization," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12553–12562, 2020.

[73] V. Piratla, P. Netrapalli, and S. Sarawagi, "Efficient domain generalization via common-specific low-rank decomposition," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 7728–7738, PMLR, 2020.

[74] K. Dohi, K. Imoto, N. Harada, D. Niizumi, Y. Koizumi, T. Nishida, H. Purohit, R. Tanabe, T. Endo, M. Yamamoto, and Y. Kawaguchi, "Description and discussion on DCASE 2022 challenge task 2: Unsupervised anomalous sound detection for machine condition monitoring applying domain generalization techniques," in *Proceedings of the 7th Detection and Classification of Acoustic Scenes and Events 2022 Workshop (DCASE2022)*, (Nancy, France), pp. 1–5, November 2022.

[75] S. Venkatesh, G. Wichern, A. Subramanian, and J. Le Roux, "Disentangled surrogate task learning for improved domain generalization in unsupervised anomalous sound detection," tech. rep., DCASE2022 Challenge, July 2022.

[76] C. Molnar, "Interpretable Machine Learning: A Guide for Making Black Box Models Explainable," 2021. Accessed: 2021-07-27.

[77] H. Lakkaraju, S. H. Bach, and J. Leskovec, "Interpretable decision sets: A joint framework for description and prediction," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 1675–1684, Association for Computing Machinery, 2016.

[78] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144, 2016.

[79] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: high-precision model-agnostic explanations," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, AAAI Press, 2018.

[80] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 3319–3328, JMLR.org, 2017.

[81] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 4768–4777, Curran Associates Inc., 2017.

[82] L. Bertossi, J. Li, M. Schleich, D. Suciu, and Z. Vagena, "Causality-based explanation of classification outcomes," in *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, DEEM '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[83] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," *CoRR*, vol. abs/1711.09784, 2017.

[84] M. Wu, M. C. Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. Doshi-Velez, "Beyond sparsity: tree regularization of deep models for interpretability," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, AAAI Press, 2018.

[85] M. Kopp *et al.*, "Anomaly Explanation with Random Forests," *Expert Systems with Applications*, vol. 149, p. 113187, 2020.

[86] E. Panjei, L. Gruenwald, E. Leal, C. Nguyen, and S. Silvia, "A survey on outlier explanations," *The VLDB Journal*, vol. 31, p. 977–1008, Jan. 2022.

[87] N. Gupta, D. Eswaran, N. Shah, L. Akoglu, and C. Faloutsos, "Beyond outlier detection: Lookout for pictorial explanation," in *Machine Learning and Knowledge Discovery in Databases* (M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, eds.), (Cham), pp. 122–138, Springer International Publishing, 2019.

[88] N. Liu, D. Shin, and X. Hu, "Contextual outlier interpretation," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 2461–2467, International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[89] J. Zhu, S. Rosset, R. Tibshirani, and T. Hastie, "1-norm support vector machines," in *Advances in Neural Information Processing Systems* (S. Thrun, L. Saul, and B. Schölkopf, eds.), vol. 16, MIT Press, 2003.

[90] H. Zhang *et al.*, "Exstream: Explaining anomalies in event stream monitoring," in *International Conference on Extending Database Technology (EDBT)*, pp. 156–167, 2017.

[91] P. Bailis *et al.*, "MacroBase: Prioritizing Attention in Fast Data," in *ACM International Conference on Management of Data (SIGMOD)*, pp. 541–556, 2017.

[92] D. Y. Yoon, N. Niu, and B. Mozafari, "Dbsherlock: A performance diagnostic tool for transactional databases," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, (New York, NY, USA), p. 1599–1614, Association for Computing Machinery, 2016.

[93] J. A. Morris and M. J. Gardner, "Statistics in medicine: Calculating confidence intervals for relative risks (odds ratios) and standardised ratios and rates," *BMJ*, vol. 296, no. 6632, pp. 1313–1316, 1988.

[94] T. A. S. Foundation, "Apache spark." `https://spark.apache.org/`. Accessed: 2024-05-31.

[95] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. J. Shenoy, "Scalla: A platform for scalable one-pass analytics using mapreduce," *ACM Transactions on Database Systems*, vol. 37, no. 4, p. 27, 2012.

[96] F. Song, Y. Diao, J. Read, A. Stiegler, and A. Bifet, "Exad: A system for explainable anomaly detection on big data traces," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 1435–1440, 2018.

[97] T. A. S. Foundation, "Monitoring and instrumentation." `https://spark.apache.org/docs/latest/monitoring.html`. Accessed: 2024-06-01.

[98] "nmon for linux." `https://nmon.sourceforge.io/pmwiki.php`. Accessed: 2024-06-01.

[99] A. Basiri *et al.*, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[100] S. Kandula *et al.*, "Detailed Diagnosis in Enterprise Networks," in *ACM SIGCOMM Conference*, pp. 243–254, 2009.

[101] V. Jacob, F. Song, A. Stiegler, B. Rad, Y. Diao, and N. Tatbul, "Exathlon: A benchmark for explainable anomaly detection over time series," *arXiv:2010.05073v3 [cs.LG]*, 2021.

[102] J. Xu, H. Wu, J. Wang, and M. Long, "Anomaly transformer: Time series anomaly detection with association discrepancy," in *International Conference on Learning Representations*, 2022.

[103] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference*, WWW '18, (Republic and Canton of Geneva, CHE), p. 187–196, International World Wide Web Conferences Steering Committee, 2018.

[104] L. Shen, Z. Li, and J. Kwok, "Timeseries anomaly detection using temporal hierarchical one-class network," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 13016–13026, Curran Associates, Inc., 2020.

[105] S. Kim, K. Choi, H.-S. Choi, B. Lee, and S. Yoon, "Towards a rigorous evaluation of time-series anomaly detection," in *AAAI Conference on Artificial Intelligence*, 2021.

[106] N. Tatbul *et al.*, "Precision and recall for time series," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[107] D. J. C. MacKay, *Probability, Entropy, and Inference*, ch. 2, pp. 27–54. Copyright Cambridge University Press, 2003.

[108] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[109] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, (USA), p. 265–283, USENIX Association, 2016.

[110] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[111] R. L. Gorsuch, *Factor Analysis*, ch. 6, pp. 143–164. John Wiley & Sons, Ltd, 2003.

[112] D. Liu, S. Alnegheimish, A. Zytek, and K. Veeramachaneni, "Mtv: Visual analytics for detecting, investigating, and annotating anomalies in multivariate time series," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, apr 2022.

[113] H. He, O. Queen, T. Koker, C. Cuevas, T. Tsiligkaridis, and M. Zitnik, "Domain adaptation for time series under feature and label shifts," 2023.

[114] J. Paparrizos, P. Boniol, T. Palpanas, R. S. Tsay, A. Elmore, and M. J. Franklin, "Volume Under the Surface: A New Accuracy Evaluation Measure for Time-Series Anomaly Detection," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2774–2787, 2022.

[115] M. Braei and S. Wagner, "Anomaly detection in univariate time-series: A survey on the state-of-the-art," *ArXiv*, vol. abs/2004.00433, 2020.

[116] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[117] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[118] D. Hawkins, *Identification of outliers*. Monographs on applied probability and statistics, London [u.a.]: Chapman and Hall, 1980.

[119] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[120] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 1278–1286, PMLR, 22–24 Jun 2014.

[121] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations*, 2016.

[122] J. M. Tomczak, "Priors in vaes." `https://jmtomczak.github.io/blog/7/7_priors.html`. Accessed: 2023-12-26.

[123] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," 2016.

[124] A. A. Alemi, B. Poole, I. Fischer, J. V. Dillon, R. A. Saurous, and K. Murphy, "Fixing a broken elbo," 2018.

[125] M. Rosca, B. Lakshminarayanan, and S. Mohamed, "Distribution matching in variational inference," 2019.

[126] M. Bauer and A. Mnih, "Resampled priors for variational autoencoders," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, vol. 89 of *Proceedings of Machine Learning Research*, pp. 66–75, PMLR, 2019.

[127] R. Groenendijk, S. Karaoglu, T. Gevers, and T. Mensink, "Multi-loss weighting with coefficient of variations," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1469–1478, 2020.

[128] A. LeNail, "Nn-svg: Publication-ready neural network architecture schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.

[129] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 785–794, Association for Computing Machinery, 2016.

[130] T. O'Malley *et al.*, "Keras Tuner." `https://github.com/keras-team/keras-tuner`, 2019.

[131] A. Bibal, R. Cardon, D. Alfter, R. Wilkens, X. Wang, T. François, and P. Watrin, "Is attention explanation? an introduction to the debate," in *Annual Meeting of the Association for Computational Linguistics*, 2022.

[132] A. Mnih, "Modern latent variable models and variational inference." `https://storage.googleapis.com/deepmind-media/UCLxDeepMind_2020/L11%20-%20UCLxDeepMind%20DL2020.pdf`, 2020. Accessed: 2024-08-08.

A Spark Streaming Dataset

This appendix provides additional details about the Spark Streaming dataset of the Exathlon benchmark. Appendix A.1 elaborates on the characteristics and settings used for the Spark Streaming applications. Appendix A.2 provides some details about the extended effect intervals set for our labeled anomalies.

A.1 Spark Streaming Applications

Our Spark workload consists of 10 stream processing applications that analyzed user click streams from the WorldCup 1998 website [95] (replicated with a scale factor for our long-running applications). The applications processed data formatted as `(user_id - - timestamp - - url)` records, using a batch interval of five seconds. Join data stored in HDFS was also available for the applications to use, in the form of a dataset called `PageRanks` containing `(page_rank - - url)` records.

We summarize the operations performed by each application below, and provide their source code under `https://github.com/exathlonbenchmark/exathlon/tree/master/apps` for more details:

App 1 (EVERY 5s, 1 BATCH): GROUP BY + COUNT
Every 5 seconds, groups the last batch by user ID and counts the number of records.

App 2 (EVERY 5s, 1 BATCH): GROUP BY + CUMULATIVE COUNT + FILTER
Every 5 seconds, groups the last batch by URL, updates a cumulative count by URL and filters out URLs with counts lower than or equal to $1,000$.

App 3 (EVERY 20s, 6 BATCHES): GROUP BY + COUNT + FILTER
Every 20 seconds (window slide), groups the last 30 seconds (i.e., the last six batches) by URL, counts the number of records, and filters out URLs with counts lower than or equal to $1,000$.

App 4 (EVERY 5s, 6 BATCHES): GROUP BY
Every 5 seconds, groups the last 30 seconds of records (i.e., the last six batches) by user ID.

App 5 (EVERY 5s, 6 BATCHES): GROUP BY + FILTER + JOIN + SUM
Every 5 seconds, groups the last 30 seconds of records (i.e., the last six batches) by user ID, filters out page URLs which are not in `PageRanks`, joins data with `PageRanks` and sums the ranks of pages by user.

App 6 (EVERY 5s, 6 BATCHES): GROUP BY + FILTER + JOIN + SUM + SAVE
Every 5 seconds, groups the last 30 seconds of records (i.e., the last six batches) by user ID, filters out page URLs which are not in `PageRanks`, joins data with `PageRanks`, sums the ranks of pages by user and saves the result to HDFS.

App 7 (EVERY 5s, 6 BATCHES): UDF + GROUP BY
Every 5 seconds, runs a user-defined function (computing the 70 first terms of the Leibniz formula for π) and groups the last 30 seconds of records (i.e., the last six batches) by user ID.

App 8 (EVERY 10s, 2 BATCHES): GROUP BY + COUNT + FILTER + SAVE
Every 10 seconds (window slide), groups the last 10 seconds of records (i.e., the last two batches) by user ID, counts the number of records and filters out user IDs with counts lower than or equal to 300.

App 9 (EVERY 10s, 2 BATCHES): GROUP BY + SAVE
Every 10 seconds (window slide), groups the last 10 seconds of records (i.e., the last two batches) by user ID and saves the result to HDFS.

App 10 (EVERY 10s, 2 BATCHES): GROUP BY + COUNT + FILTER + SAVE
Every 10 seconds (window slide), groups the last 10 seconds of records (i.e., the last two batches) by URL, counts the number of records, filters out URLs with counts lower than or equal to 500 and saves the result to HDFS.

The operation results were therefore entirely saved to HDFS for applications 6, 8, 9 and 10. For other applications, the results were only saved to HDFS for the first batch. Overall, these applications commonly involved group-by operations, and varied in terms of windowing, parameters and filtering conditions. Two of the applications additionally involved join operations, and one of them used a user-defined function. As such, our workload aimed to capture a diverse mix of popular streaming primitives.

We note that the batch interval, window size, and window slide parameters were always kept constant across the runs of a given application. These application-specific parameters led to important characteristics for the corresponding traces, which we refer to as their *processing period* and *periodic volume*. The processing period corresponds to the time elapsed between two consecutive computations triggered by the application, marking the consideration of new input records to process. For non-windowed applications, this corresponds to the batch interval, which was always set to five seconds. For windowed applications, this corresponds to the window slide parameter, set to 20 seconds for application 3, and 10 seconds for applications 8, 9, and 10. The periodic volume refers to the volume of data periodically processed by the application (i.e., every "processing period"). This volume depends on both the number of batches considered by the application (indicated above) and the data sender input rate, defining the number of records sent per second for the application to process.

A.1.1 Spark Settings

Throughout data collection, a same application could be launched with different Spark *settings*, which were reflected in various ways in the recorded time series. Some setting values were correlated with, although not completely determined by, the *type* of trace recorded (i.e., undisturbed or one of the five disturbed types), since introducing different types of anomalous events could require different settings for data collection to work properly. This section goes over the main "setting-related" characteristics that were observed to have an impact in the traces data. In this work, we define the overall "settings" used for a given trace as the values taken by each of these differentiating factors.

Number of Active Executors

The first differentiating factor across traces of a same application could be its number of active Spark executors. We include this aspect as a general settings component here, although its impact was likely reduced by the "active executor averaging" of our feature engineering described in Section 4.2.4. We detail the number of active executors for the traces of each type below:

Undisturbed
Always 2 active executors.

T1 (Bursty Input)
Always 2 active executors, except for trace `4_1_100000_61`, which has 3.

T2 (Bursty Input Until Crash)
Always 3 active executors.

T3 (Stalled Input)
Always 2 active executors.

T4 (CPU Contention)
Always 2 active executors, except for traces `3_4_1000000_81` and `9_4_1000000_78`, which have 3.

T5 (Process Failure)
Always 3 active executors, except for traces `5_5_1500000_92` and `10_5_1000000_85`, which have 2.

"Memory Profile"

We define the second settings component of a trace broadly as its "memory profile", covering memory-related settings that varied together for a given trace. These metrics include the maximum memory set for the driver's block manager, as well as the maximum memory set for the executors' JVM and garbage collection (PS-Eden-Space, PS-Old-Gen and PS-Survivor-Space). All these settings have a direct impact on the maximum memory and memory usage metrics of the corresponding traces, besides indirectly impacting other metrics as well. We detail the maximum memory set for the executors JVM for the traces of each type below:

Undisturbed
Always 30GB, except for trace `9_0_100000_3`, for which it is 9GB.

T1 (Bursty Input)
Always 9GB, except for traces `5_1_500000_62` and `6_1_500000_65`, for which it is 15GB.

T2 (Bursty Input Until Crash), T4 (CPU Contention), T5 (Process Failure)
Always 15GB, except for traces `2_5_1000000_87` and `5_5_1000000_91`, for which it is 9GB.

T3 (Stalled Input)
Always 15GB, except for trace `6_3_200000_76`, for which it is 30GB.

A.2 Extended Effect Intervals

This section describes the way we set the extended effect interval (see Table 3.2) for each anomalous event. Each anomaly instance was initially labeled with its known type and *root cause interval* (RCI). However, we observed that some injected events could have a long-lasting effect on relevant metrics, such as the scheduling delay and processing delay of the last completed batch, even after their end time. To characterize such events, we therefore sought to extend their root cause interval to include this long-lasting effect. More specifically, we used domain knowledge to set an *extended effect interval* (EEI) for each anomalous event, starting immediately after the end of its RCI, and ending at a point after which we deem anomaly detection not helpful. In practice, we either set the end of an EEI to when the application had fully restarted, or to when its main metrics had come back to normal. Here are the rules we used for each type of anomalous event:

- **Bursty Input (Type 1).** The end of the EEI is set to the point when highly related metrics, such as the scheduling delay and processing delay of the last completed batch, come back to normal.
- **Bursty Input Until Crash (Type 2).** The EEI is set to null, because the root cause event already ends at the time of the application crash.
- **Stalled Input (Type 3).** The end of the EEI is set to the point when the processing delay of the last completed batch comes back to normal (the application restarts processing data at its usual rate).

- **CPU Contention (Type 4).** If the effect of the event is just an increased processing time, we set the end of the EEI to the time when the scheduling delay and processing delay of the last completed batch come back to normal. If its effect is an application crash, we set the end of the EEI to the time when the application restarts (typically one minute after the crash in practice).
- **Driver Failure (Type 5).** The end of the EEI is set to when the application restarts.
- **Executor Failure (Type 6).** The end of the EEI is set to the point when the scheduling delay of the last completed batch comes back to normal.

B VAE Framework

This appendix provides a brief reminder of the Variational Autoencoder (VAE) framework [119, 120] used in DIVAD (Appendix B.1), based on some content from [132], as well as the adaptation of this framework to DIVAD's dependency structure (Appendix B.2).

B.1 Single Latent Variable

Let $p_{\boldsymbol{\theta}}(\mathbf{x})$ be a latent variable model considering a single continuous latent variable $\mathbf{z}$, with $\boldsymbol{\theta} \in \Theta$ a vector of **model parameters**. We model the marginal likelihood of the observed variable $\mathbf{x}$ as follows:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z},$$

with $p(\mathbf{z})$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ two tractable distributions.

Our goal is then to find $\hat{\boldsymbol{\theta}}$ that best fits the training observations $\{\boldsymbol{x}_i\}_{i=1}^N$ using maximum likelihood estimation:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$
$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \log \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\mathbf{z})p(\mathbf{z})d\mathbf{z}.$$

We can see that, for each observation $\boldsymbol{x}_i$, this maximization process requires estimating an integral over the continuous latent variable $\mathbf{z}$, which is intractable in the general case.

B.1.1 Variational Inference

One way to estimate the log-likelihood and its gradient in a tractable way is through *variational inference*, considering a tractable distribution $q_i(\mathbf{z})$ for each instance $\boldsymbol{x}_i$ (e.g., $q_i(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_i, \sigma_i^2 \boldsymbol{I})$). For *each* observation $\boldsymbol{x}_i$, we have for *any* such $q_i(\mathbf{z})$ (as long as it is non-zero over the support of $p(\mathbf{z})$):

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) &= \log \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\mathbf{z})p(\mathbf{z})d\mathbf{z} \\
&= \log \int q_i(\mathbf{z}) \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\mathbf{z})p(\mathbf{z})}{q_i(\mathbf{z})} d\mathbf{z} \\
&= \log \mathbb{E}_{q_i(\mathbf{z})} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\mathbf{z})p(\mathbf{z})}{q_i(\mathbf{z})} \\
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) &\leq \mathbb{E}_{q_i(\mathbf{z})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\mathbf{z})p(\mathbf{z})}{q_i(\mathbf{z})} \quad , \quad \text{using Jensen's inequality.}
\end{aligned}$$

By using any such $q_i(\mathbf{z})$, we can therefore get a lower bound on the marginal log-likelihood of a particular instance $\boldsymbol{x}_i$:

$$\mathcal{L}^{(i)}(\boldsymbol{x}_i;\boldsymbol{\theta}) := \mathbb{E}_{q_i(\mathbf{z})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\mathbf{z})}{q_i(\mathbf{z})}.$$

A good choice of $q_i(\mathbf{z})$ for a tight lower bound is to set it as an approximation of the "true" posterior estimate of the data for the instance, $p_\theta(\mathbf{z}|\boldsymbol{x}_i)$ (itself intractable). This definition of $q_i(\mathbf{z})$ is called the *variational posterior* $q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)$ (with $\boldsymbol{\phi}_i \in \Phi$ observation-specific **variational parameters**). The corresponding lower bound is called the evidence lower-bound (ELBO):

$$\begin{aligned}
\mathcal{L}^{(i)}_{\text{ELBO}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\phi}_i) &:= \mathbb{E}_{q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\mathbf{z})}{q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)} \log \frac{p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}_i)p_{\boldsymbol{\theta}}(\boldsymbol{x}_i)}{q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - D_{\text{KL}}(q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)\|p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}_i)) \\
\mathcal{L}^{(i)}_{\text{ELBO}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\phi}_i) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - D_{\text{KL}}(q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)\|p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}_i)).
\end{aligned}$$

As we can see, maximizing the ELBO, a *tractable* entity, for a given observation $\boldsymbol{x}_i$ therefore has an effect on two *intractable* entities: the marginal log-likelihood of the observation and the KL divergence between variational and true posterior distributions (called the *variational gap*). On the one hand, maximizing the ELBO with respect to $\boldsymbol{\phi}_i$ reduces the variational gap, making the variational posterior better estimate the true posterior. Importantly, this also has the effect of making the ELBO itself better approximate the true marginal log-likelihood of the observation, hence tightening the bound. Indeed, we have:

$$D_{\text{KL}}(q_{\boldsymbol{\phi}_i}(\mathbf{z}|\boldsymbol{x}_i)\|p_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}_i)) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \mathcal{L}^{(i)}_{\text{ELBO}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\phi}_i).$$

On the other hand, maximizing the ELBO with respect to $\boldsymbol{\theta}$ has the effect of both maximizing the marginal log-likelihood of the observation (our main objective), as well as reducing the variational gap.

For this reason, the maximization process of the ELBO can be seen as alternating between 1) *making this bound a better approximation of the true marginal log-likelihood*, and 2) *maximizing the true marginal log-likelihood itself.*

B.1.2 Amortized Variational Inference

In the current setting, we consider a different ELBO and variational posterior optimization for every observation $\boldsymbol{x}_i$ (whether it is a training or a test one). Doing this is typically inefficient for large datasets, and also because it does not allow the model to share any knowledge across instances.

Amortized variational inference seeks to amortize the cost of this separate optimization procedure, by replacing it with a functional approximation: instead of optimizing the variational parameters $\boldsymbol{\phi}_i$ for every observations, we train a neural network, called the **inference network**, to *predict* them from the observation. In this context, $\boldsymbol{\phi}$ does not refer to the variational parameters anymore, but to the parameters of the inference network used to predict them. The updated ELBO therefore reads:

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\phi}) := \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)},$$

where $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)$ is typically specified as $\mathcal{N}(\text{NN}_{\boldsymbol{\phi}}(\boldsymbol{x}_i), \text{NN}_{\boldsymbol{\phi}}(\boldsymbol{x}_i))$, with $\text{NN}_{\boldsymbol{\phi}}$ the inference network.

B.1.3 ELBO Maximization (Reparameterization Trick)

We compute the gradient of the ELBO with respect to the model parameters $\boldsymbol{\theta}$ using Monte Carlo sampling from the variational posterior (here using K samples):

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\phi}) &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)}\left[\log\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)}\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)}[\nabla_{\boldsymbol{\theta}}\log p_\theta(\boldsymbol{x}_i,\mathbf{z})] \\ \nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\phi}) &\approx \frac{1}{K}\sum_{k=1}^{K}\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\mathbf{z}^{(k)}) \quad , \quad \text{with} \quad \mathbf{z}^{(k)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i).\end{aligned}$$

However, computing the gradient of the ELBO with respect to the variational parameters $\boldsymbol{\phi}$ is harder, due to the sampling procedure also depending on $\boldsymbol{\phi}$:

$$\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)}\left[\log\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)}\right].$$

To compute this gradient, we use the *reparameterization trick*, that is, reparameterizing samples $\mathbf{z}$ as functions of other samples ϵ coming from a parameter-free distribution $p(\epsilon)$:

$$\mathbf{z} = g(\epsilon;\boldsymbol{\phi}),$$

hence factoring out the sampling procedure from the variational parameters. As long as $g(\epsilon,\boldsymbol{\phi})$ is differentiable with respect to $\boldsymbol{\phi}$, we then have:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z})}[f(\mathbf{z})] &= \nabla_{\boldsymbol{\phi}}\mathbb{E}_{p(\epsilon)}[f(g(\epsilon,\boldsymbol{\phi}))] & \\ &= \mathbb{E}_{p(\epsilon)}[\nabla_{\boldsymbol{\phi}}f(g(\epsilon,\boldsymbol{\phi}))] & p(\epsilon) \text{ does not depend on } \boldsymbol{\phi} \\ \nabla_{\boldsymbol{\phi}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z})}[f(\mathbf{z})] &= \mathbb{E}_{p(\epsilon)}[\nabla_{\mathbf{z}}f(\mathbf{z})|_{\mathbf{z}=g(\epsilon,\phi)}\nabla_{\phi}g(\epsilon,\phi)] & \text{Using the chain rule.}\end{aligned}$$

For $\mathrm{z}\sim\mathcal{N}(\mu,\sigma^2)$, this amounts to setting:

$$\mathrm{z} = \mu + \sigma\epsilon \quad \text{with } \epsilon\sim\mathcal{N}(0,1).$$

B.1.4 Variational Autoencoder (VAE) Framework

Variational autoencoders are generative models with continuous latent variables, where both the likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\mathbf{z})$ and the variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x}_i)$ are parameterized using neural networks. In this setting, similarly to the variational posterior, the parameters $\boldsymbol{\theta}$ of the likelihood do not refer to the model parameters anymore, but to the parameters of the **decoder network** used to predict them. The inference network is also called the **encoder network**. Relevant components are therefore:

- The encoder $\text{NN}_{\boldsymbol{\phi}}$, computing the variational posterior $q_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\text{NN}_{\boldsymbol{\phi}}(\mathbf{x}), \text{NN}_{\boldsymbol{\phi}}(\mathbf{x}))$ from $\mathbf{x}$.
- The decoder $\text{NN}_{\boldsymbol{\theta}}$, computing the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\text{NN}_{\boldsymbol{\theta}}(\mathbf{z}), \text{NN}_{\boldsymbol{\theta}}(\mathbf{z}))$ from $\mathbf{z}$.
- The prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

They are trained by maximizing the ELBO, expressed as:

$$\mathcal{L}_{\text{ELBO}}(\mathbf{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})),$$

using amortized variational inference and the reparametrization trick.

B.2 Adaptation to DIVAD's Dependency Structure

Following the dependency structure specified in Section 5.2, the marginal log-likelihood of the observed variable $\mathbf{x}$ given its domain d reads:

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathrm{d}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_y, \mathbf{z}_d) p_{\boldsymbol{\theta}}(\mathbf{z}_d|\mathrm{d}) p(\mathbf{z}_y) d\mathbf{z}_d d\mathbf{z}_y,$$

which, using amortized inference, corresponds to the following ELBO to maximize:

$$\mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{x}_i, d_i; \boldsymbol{\theta}_{yd}, \boldsymbol{\theta}_d, \boldsymbol{\phi}_d, \boldsymbol{\phi}_y) = \mathbb{E}_{q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x}_i) q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i)} \log \frac{p_{\boldsymbol{\theta}_{yd}}(\boldsymbol{x}_i|\mathbf{z}_y, \mathbf{z}_d) p_{\boldsymbol{\theta}_d}(\mathbf{z}_d|d_i) p(\mathbf{z}_y)}{q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x}_i) q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i)},$$

with d_i the domain of $\boldsymbol{x}_i$. This corresponds to the ELBO of Equation 5.1 (without its KL divergence terms weighted by a factor β):

$$\begin{aligned}\mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{x}_i, d_i; \boldsymbol{\theta}_{yd}, \boldsymbol{\theta}_d, \boldsymbol{\phi}_d, \boldsymbol{\phi}_y) = \mathbb{E}_{q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x}_i) q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i)} [\log p_{\boldsymbol{\theta}_{yd}}(\boldsymbol{x}_i|\mathbf{z}_d, \mathbf{z}_y)] \\ - D_{\mathrm{KL}}(q_{\boldsymbol{\phi}_y}(\mathbf{z}_y|\boldsymbol{x}_i) \| p(\mathbf{z}_y)) - D_{\mathrm{KL}}(q_{\boldsymbol{\phi}_d}(\mathbf{z}_d|\boldsymbol{x}_i) \| p_{\boldsymbol{\theta}_d}(\mathbf{z}_d|d_i)).\end{aligned}$$

ECOLE
DOCTORALE

Titre : Détection d'Anomalies dans les Séries Temporelles de Grande Dimension sur des Domaines Hétérogènes

Mots clés : Détection d'Anomalies, Séries Temporelles, Généralisation de Domaine, Modélisation Générative, Apprentissage Contrastif

Résumé : L'adoption généralisée des services numériques, ainsi que leur échelle et complexité, a rendu les incidents dans les opérations informatiques de plus en plus probables, diversifiés et impactants pour les entreprises, créant le besoin de solutions automatisées pour les prévenir. Pour répondre à ce besoin, cette thèse se concentre sur le problème de la détection d'anomalies dans les séries temporelles de grande dimension, dans le cadre de l'"Intelligence Artificielle pour les Opérations Informatiques" (AIOps).
Cette thèse propose tout d'abord de nouveaux outils de benchmarking pour la détection d'anomalies explicable dans des séries temporelles de grande dimension. L'outil principal est Exathlon, comprenant (i) un jeu de données provenant de la surveillance d'applications Spark Streaming, (ii) un framework d'évaluation des méthodes de détection d'anomalies et de "découverte d'explications", et (iii) une pipeline complète de détection d'anomalies explicable.
Cette thèse propose ensuite une analyse de benchmark de méthodes de détection d'anomalies non supervisées, révèlant leur performance limitée avec trois limitations principales : (L1) une vulnérabilité au changement de comportement normal entre les données d'entraînement et de test, (L2) une production de faux négatifs pour les anomalies les plus complexes, et (L3) une production de faux positifs pour les motifs de "bruit" normaux dans les données de test.
La partie suivante traite explicitement de L1 via une nouvelle méthode de généralisation de domaine appelée DIVAD, définissant les différents contextes de fonctionnement normal comme des "domaines hétérogènes", et associant le changement de comportement normal à un "changement de domaine". Nos expériences montrent que DIVAD est particulièrement efficace pour traiter L1, améliorant ainsi la performance jusqu'à 20%, tout en étant applicable au-delà des données d'Exathlon.
Afin de traiter conjointement L1-3, cette thèse propose enfin de nouvelles méthodes contrastives faiblement supervisées, dont la principale est CEADAL. Nos expériences montrent l'efficacité de CEADAL pour traiter L1-3, atteignant les scores F1 maximal et médian les plus élevés des méthodes comparées.

Title : High-Dimensional Time Series Anomaly Detection across Heterogeneous Domains

Keywords : Anomaly Detection, Time Series, Domain Generalization, Generative Modeling, Contrastive Learning

Abstract : The widespread adoption of digital services, along with their scale and complexity, has made incidents in IT operations increasingly likely, diverse and impactful for companies, creating the need for automated methods to prevent them. To respond to such needs, this thesis focuses on the problem of anomaly detection in high-dimensional time series, within "Artificial Intelligence for IT Operations" (AIOps).
This thesis starts by contributing new benchmarking tools for explainable anomaly detection in high-dimensional time series. The main tool is Exathlon, consisting of (i) a dataset centered around a Spark Streaming application monitoring use case, (ii) an evaluation framework to assess anomaly detection and explanation discovery methods, and (iii) an end-to-end explainable anomaly detection pipeline.
This thesis next contributes a benchmarking analysis of unsupervised anomaly detection methods, revealing their limited performance with three main limitations: (L1) a vulnerability to normal behavior shift from the training to the test data, (L2) a production of false negatives for the hardest anomalies, and (L3) a production of false positives for normal but "noisy" patterns in test data.
The next part explicitly addresses L1 with a new domain generalization method called DIVAD, defining the different contexts of normal operation as "heterogeneous domains", and associating normal behavior shift to the concept of "domain shift". Our experiments show that DIVAD is particularly effective in addressing L1, thus improving performance by up to 20%, while also being applicable beyond Exathlon's data.
To jointly address L1-3, this thesis finally contributes new weakly-supervised contrastive methods, the main one being CEADAL. Our experiments show the effectiveness of CEADAL in addressing L1-3, achieving the highest maximum and median peak F1-scores of the methods compared.

Institut Polytechnique de Paris
91120 Palaiseau, France