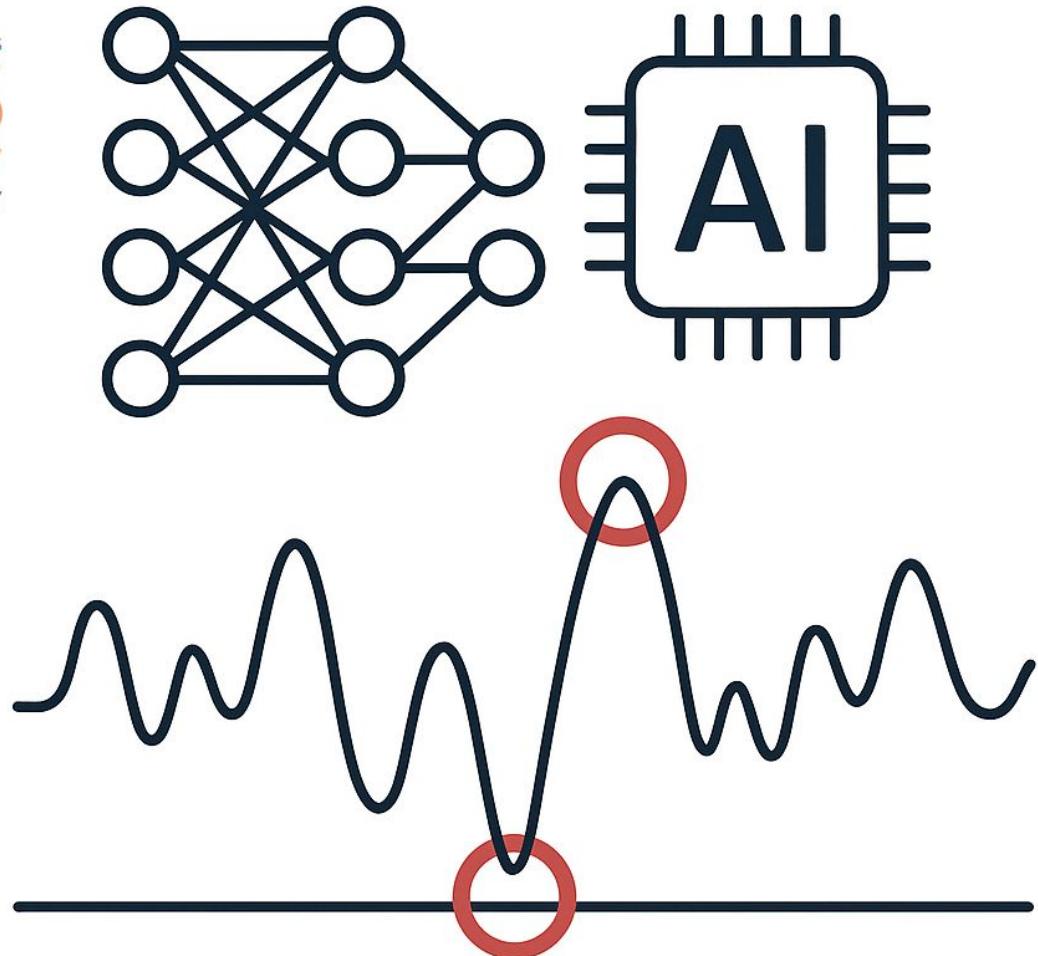




INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS
IJCNN2025
30 JUNE - 5 JULY 2025 | ROME, ITALY
INTERNATIONAL NEURAL NETWORK SOCIETY

Continual Anomaly Detection Tutorial





Roberto Corizzo

Assistant Professor
Department of Computer Science



rcorizzo@american.edu

rcorizzo.com

[@robcorizzo](https://twitter.com/robcorizzo)



Kamil Faber

Research Associate
Department of Computer Science



kfaber@agh.edu.pl

<https://scholar.google.com/citations?user=5oJ30hkAAAAJ&hl=en>

Tutorial Website

<https://github.com/lifelonglab/ijcnn-2025-tutorial/>



Preliminaries: Anomaly Detection

Anomaly detection

- The process of identifying data that represents a deviation from the normal conditions.
- Paramount importance in multiple fields, such as detection of intrusions in cybersecurity and detection of defects in manufacturing process.

Anomaly Detection

Offline (batch)

Online (adaptation)

Anomaly detection



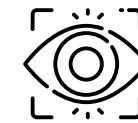
Applications

Cybersecurity
Healthcare
Industrial monitoring



Normal behavior

Limited access to
data about anomalies



Detection

Identification of
deviations from
normal

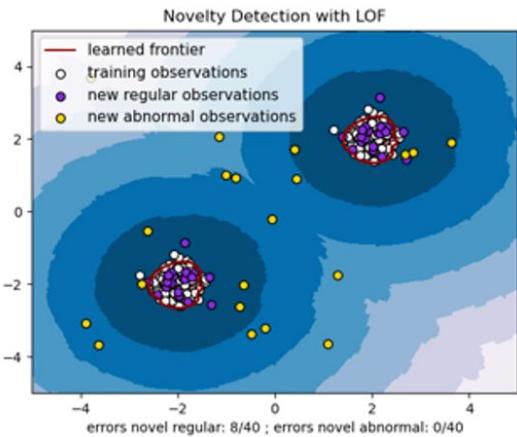
Semi-supervised learning setting

- Many real-world applications are characterized by imbalanced data
- When the imbalance is extreme, classifiers may struggle to properly represent minority classes
- This is particularly true in anomaly detection settings
 - Credit card transactions
 - Manufacturing processes
 - Sensor data

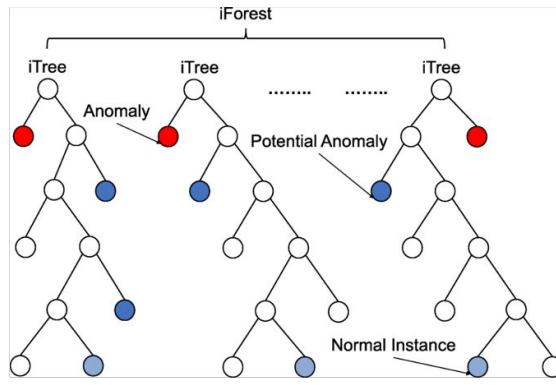
Semi-supervised learning setting

- Can we give up on modeling minority classes in the classifier?
 - Instead of differentiating between positive and negative:
 - Train a model to learn the distribution of the normal behavior
 - For which data is abundant
 - Classify unseen data points as normal/anomaly using a recognition-based approach:
 - How distant is the new data point from the model representation of the normal class?

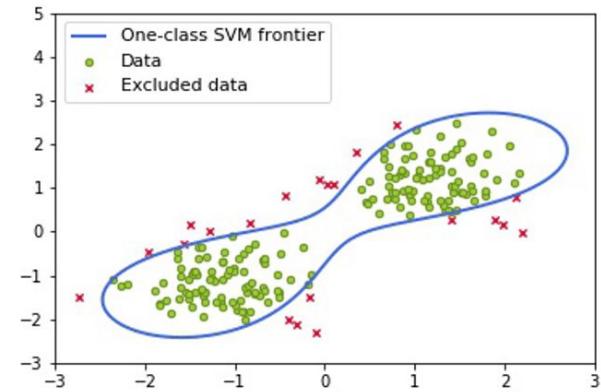
One-Class Models



**Local Outlier Factor
(LOF)**



Isolation Forest



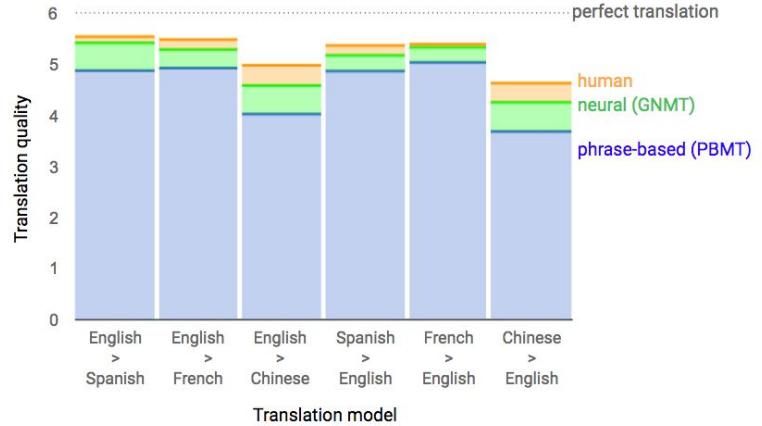
**One Class SVM
(OCSVM)**

Python Libraries

- **SkLearn**
 - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html>
 - <https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html>
 - <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html>
- **Keras/Tensorflow**
 - Define your own autoencoder models
- **PyOD**
 - <https://pyod.readthedocs.io/en/latest/>
 - Established in 2017, has become a go-to Python library for detecting anomalous/outlying objects in multivariate data.
 - PyOD includes more than 50 detection algorithms, from classical LOF (SIGMOD 2000) to the cutting-edge ECOD and DIF (TKDE 2022 and 2023).

Preliminaries: Continual Learning

Landscape of current ML/DL methods



Limitations

- Classical ML/AI systems are **limited** to performing tasks for which they have been specifically programmed and trained.
- They are **inherently unreliable** when encountering different situations.
- This is an issue in applications where **situations can be unpredictable** and the ability to **react quickly** and **adapt to dynamic circumstances** is of primary importance.

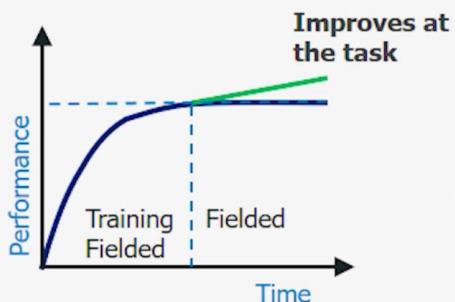
Lifelong Machine Learning (or L2M) considers systems that can learn many tasks over a lifetime from one or more domains.

They efficiently and effectively retain the knowledge they have learned and use that knowledge to more efficiently and effectively learn new tasks.

Lifelong Learning desiderata for AI

L2M is concerned with learning machines that will improve their performance over their lifetimes

Continuously Improve Performance



Current ML based on large datasets; data may be scarce

Adapt to New Conditions

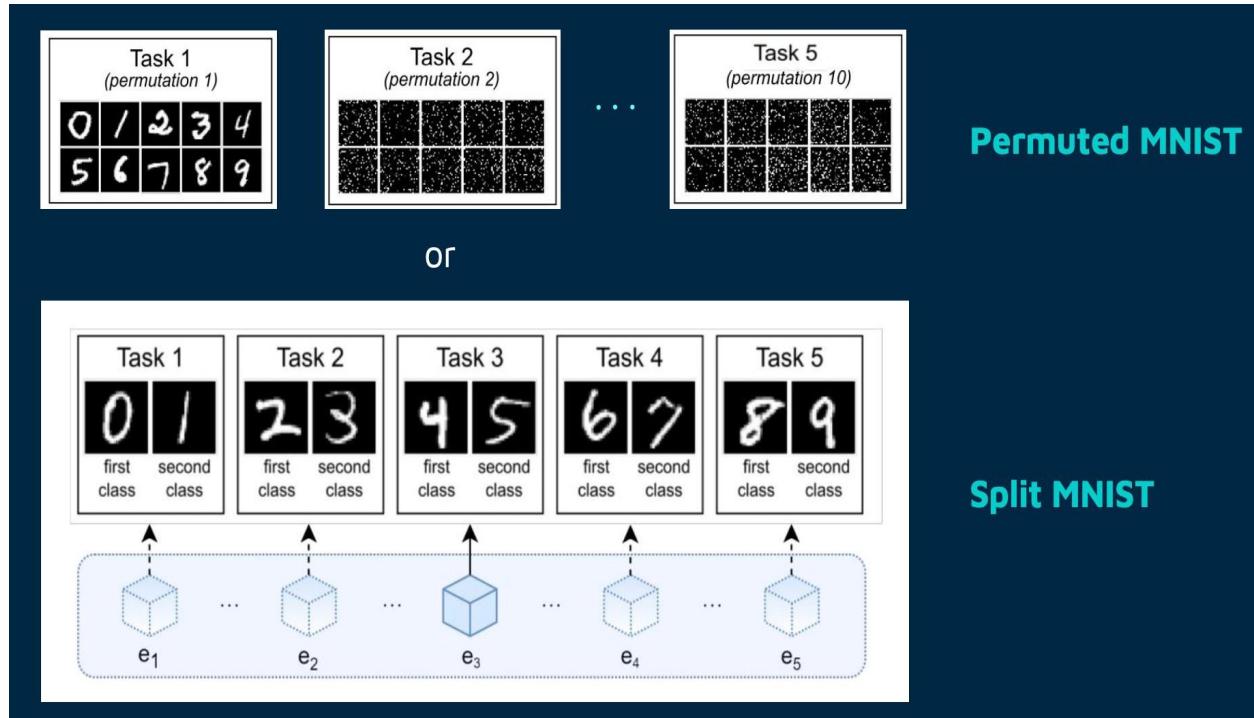


Situation may change after training and fielding (external, internal)

Core Capabilities of a L2M System

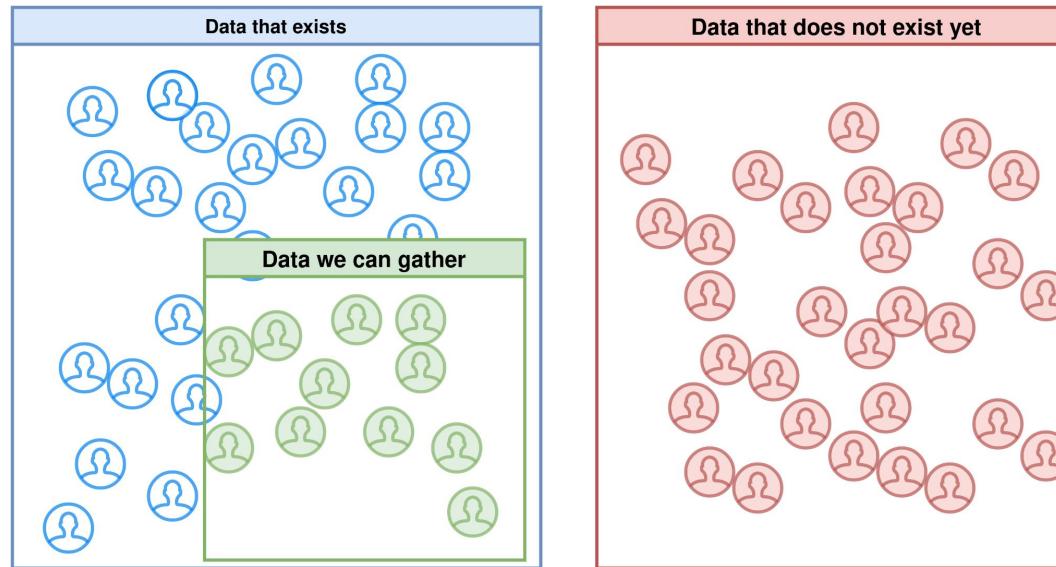
1. Continual learning
2. Adaptation to new tasks and circumstances
3. Goal-driven perception
4. Selective plasticity
5. Safety and monitoring

Lifelong Learning: A simplistic scenario



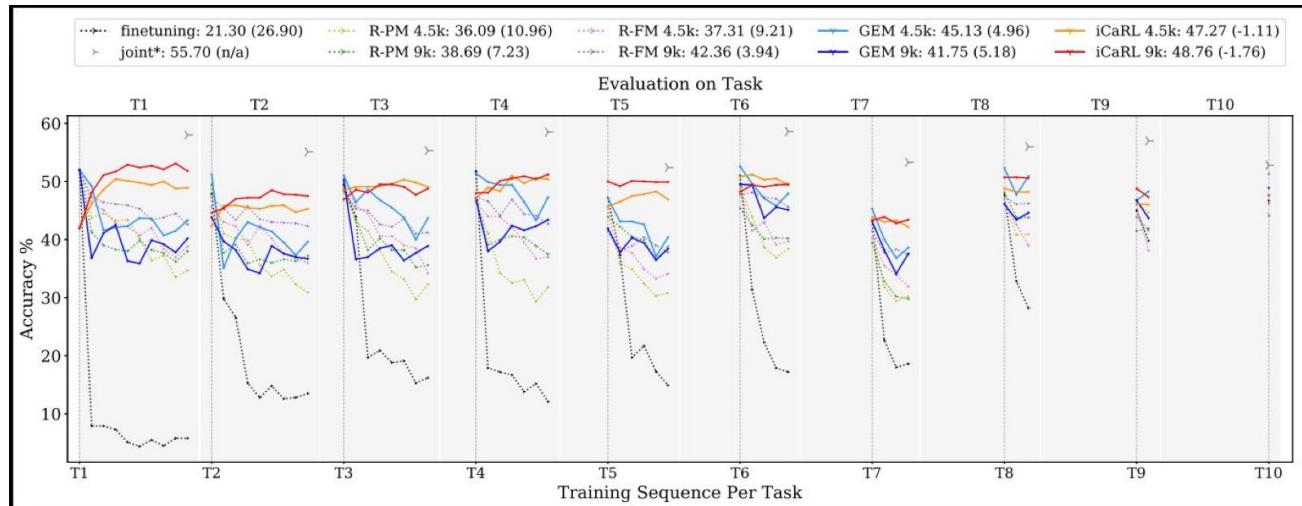
Source: ContinualAI

Lifelong Learning: A simplistic scenario



Lifelong Learning and catastrophic forgetting

Catastrophic forgetting is the tendency of a model to completely and abruptly forget previously learned information upon learning new information. Mostly due to Gradient Descent.



The Stability-Plasticity Dilemma:

- Remember past concepts
- Learn new concepts
- Generalize

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., ... & Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7), 3366-3385.

Outline

1. Motivation for continual anomaly detection
2. Challenges of anomaly detection in continual learning
3. Scenarios, metrics, and strategies
4. pyCLAD: A universal framework for continual lifelong anomaly detection

Outline

1. Motivation for continual anomaly detection
2. Challenges of anomaly detection in continual learning
3. Scenarios, metrics, and strategies
4. pyCLAD: A universal framework for continual lifelong anomaly detection

Open gaps in state-of-the-art



Anomaly detection

- ✓ Single training
- ✓ Online learning with forgetting

[1] Zamanzadeh Darban, Zahra, et al. "Deep learning for time series anomaly detection: A survey." ACM Computing Surveys 57.1 (2024): 1-42.

[2] Aggarwal, Charu C., and Charu C. Aggarwal. An introduction to outlier analysis. Springer International Publishing, 2017.

[3] Ruff, Lukas, et al. "A unifying review of deep and shallow anomaly detection." Proceedings of the IEEE 109.5 (2021): 756-795.

Open gaps in state-of-the-art



Continual learning

- ✓ Image classification
- ✓ Object recognition
- ✓ Reinforcement Learning

- [1] Wang, Liyuan, et al. "A comprehensive survey of continual learning: Theory, method and application." *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).
- [2] Parisi, German I., et al. "Continual lifelong learning with neural networks: A review." *Neural networks* 113 (2019): 54-71.
- [3] Mitchell, Rupert, et al. "Continual Learning Should Move Beyond Incremental Classification." *arXiv preprint arXiv:2502.11927* (2025).

Open gaps in state-of-the-art



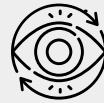
Anomaly detection

- ✓ Single training
- ✓ Online learning with forgetting
- Continual learning



Continual learning

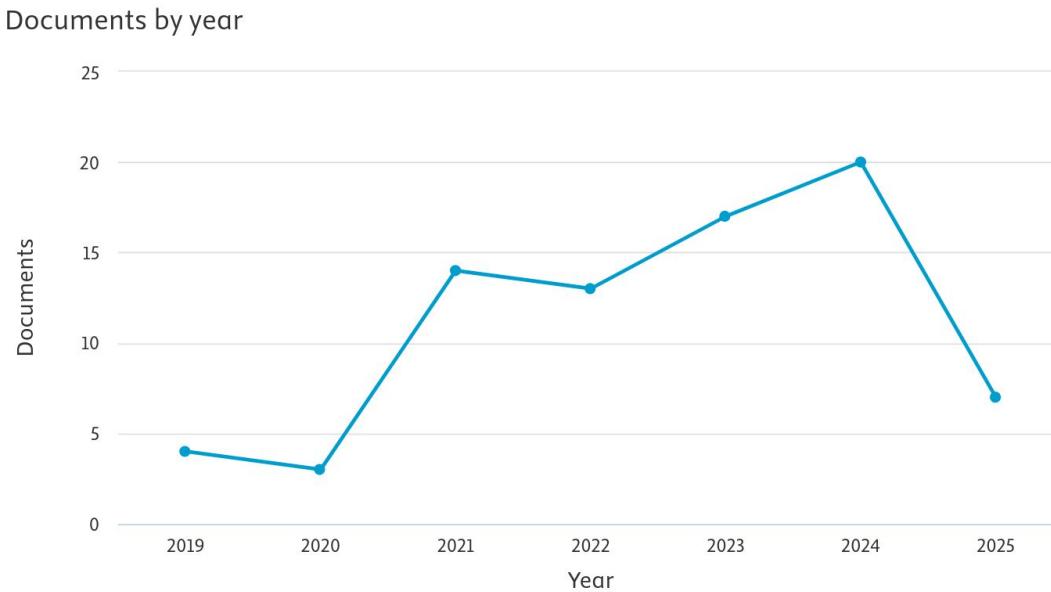
- ✓ Image classification
- ✓ Object recognition
- ✓ Reinforcement Learning
- Anomaly detection



Continual anomaly detection

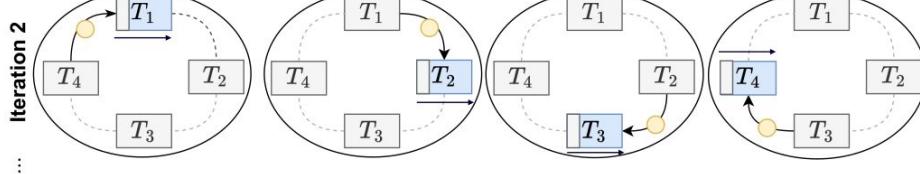
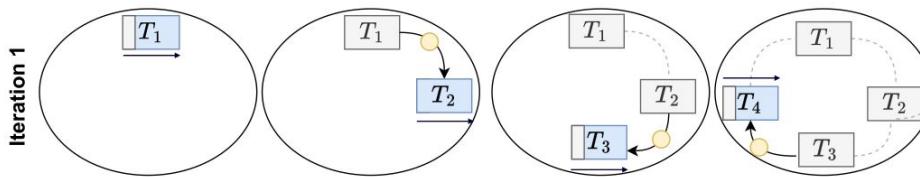
- Anomaly detection models are often embedded in evolving environments
- Opportunity to create methods more robust to real life conditions
- Unique challenges

Continual anomaly detection works



((TITLE-ABS-KEY ("continual learning" OR "lifelong learning") AND TITLE-ABS-KEY ("anomaly detection"))
OR (TITLE-ABS-KEY ("continual anomaly detection") OR TITLE-ABS-KEY ("lifelong anomaly detection")))
AND (LIMIT-TO (SUBJAREA , "COMP"))

Non-Lifelong Anomaly Detection

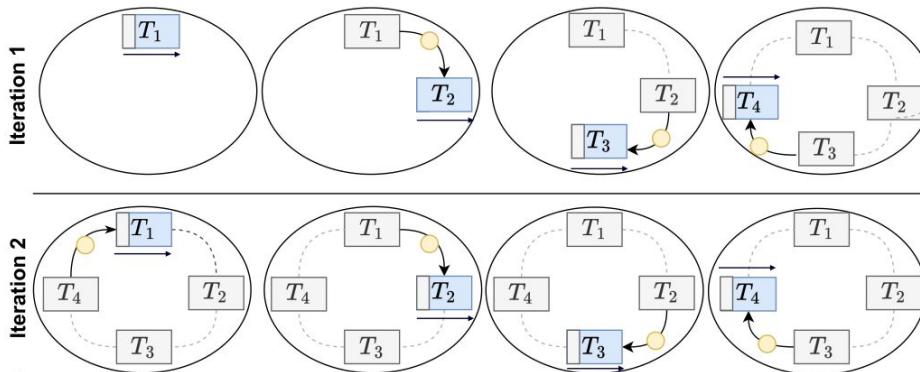


A scenario with four recurring tasks T1, T2, T3, T4.

Conventional anomaly detection requires constant model updates and results in detection delays.



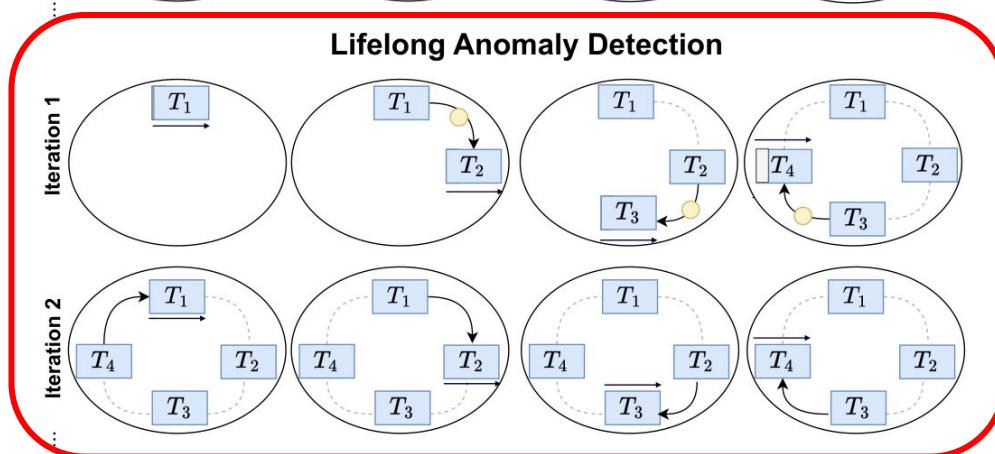
Non-Lifelong Anomaly Detection



A scenario with four recurring tasks T₁, T₂, T₃, T₄.

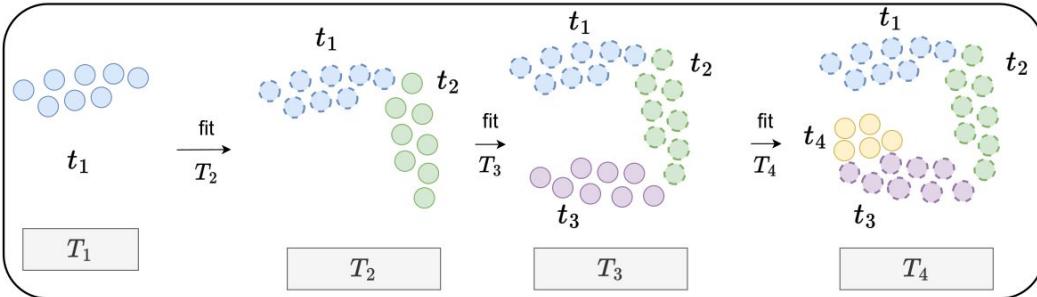
Conventional anomaly detection requires constant model updates and results in detection delays.

Lifelong learning mitigates this burden by retaining knowledge of tasks



- Model has knowledge of this task □ Model has no knowledge of this task
- Delay deriving from model update → Current task
- ↔ Model is being updated

Model training/update as new data is received



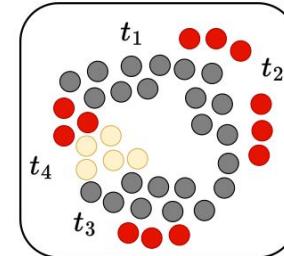
● ● ● Fading/forgotten task

$T_1 \dots T_n$ Set of tasks that the model is able to deal with

● ● ● Normal data local to a task (incorporated by the model)

Legend: Initial model training and update

Example of Inference with normal + anomaly data (all tasks)



● ● ● Normal data correctly classified as normal (without task identification)

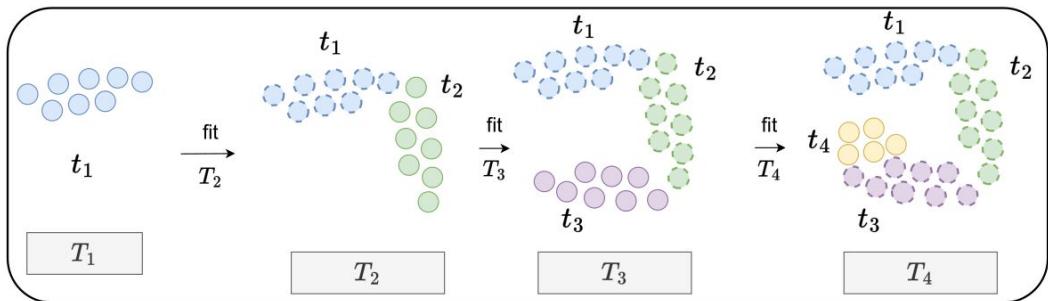
● Correctly classified anomalies

● Normal data misclassified as anomaly

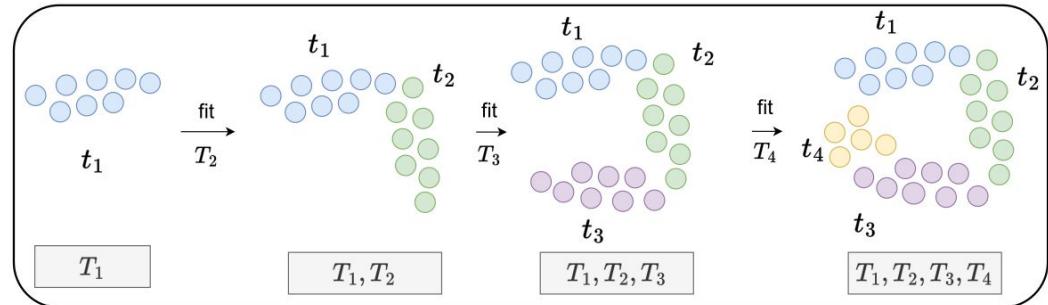
Legend: Inference

Comparison of training/update and inference for non-lifelong and lifelong anomaly detection in the scenario with four tasks T_1, T_2, T_3, T_4

Non-lifelong Anomaly Detection
Adaptation -> Forgetting



Lifelong Anomaly Detection
Adaptation + Knowledge retention



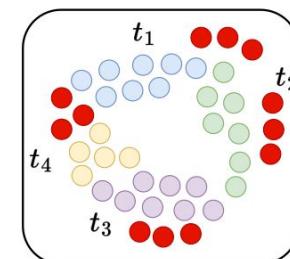
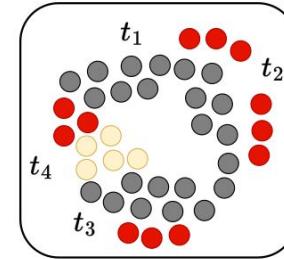
● ● ● Fading/forgotten task

$T_1 \dots T_n$ Set of tasks that the model is able to deal with

● ● ● Normal data local to a task (incorporated by the model)

Legend: Initial model training and update

Example of Inference with normal + anomaly data (all tasks)



● ● ● Normal data correctly classified as normal (without task identification)

● Correctly classified anomalies

● Normal data misclassified as anomaly

Legend: Inference

Comparison of training/update and inference for non-lifelong and lifelong anomaly detection in the scenario with four tasks T_1, T_2, T_3, T_4

Takeaways

- Conventional (non-continual) anomaly detection leads to model that:
 - Only adapt to the new normal class distribution -> **forgetting past knowledge**
 - Are unable to **leverage past skills** or combine them with recent skills
 - Are evaluated in **simplistic experimental settings**
 - Require significant **computational resources**
 - Do not **leverage task similarity** and **knowledge transfer** across tasks
 - Lack of **comprehensive view** of the environment
- These limitations provide the motivation for continual anomaly detection

Questions?



Outline

1. Motivation for continual anomaly detection
2. Challenges of anomaly detection in continual learning
3. Scenarios, metrics, and strategies
4. pyCLAD: A universal framework for continual lifelong anomaly detection

Challenge I :

Limited availability of anomalies compared to normal data (class imbalance)

Kaggle Credit Card Fraud

- **Domain:** Finance
- **Anomaly Class:** Fraudulent transactions
- **Anomaly Percentage:** 0.17% anomalies
- **Description:** Out of 284,807 transactions, only 492 are fraudulent.

NASA Turbofan Engine Degradation

- **Domain:** Industrial Equipment Monitoring
- **Anomaly Class:** Engine failure cycles
- **Anomaly Percentage:** <1%
- **Description:** The majority of operational cycles show no degradation until close to failure.

Rare Event Prediction in Aerospace (Airline Incident Reports)

- **Domain:** Transportation Safety
- **Anomaly Class:** Incidents/Failures
- **Anomaly Percentage:** <<0.1%
- **Description:** Incidents such as near misses or mechanical failures are very rare compared to normal flights.

Challenge II:

Possible lack of information about task changes and task identities
(task-incremental vs. task-agnostic).

| Aspect | Image Classification | Anomaly Detection |
|----------------------------|--|--|
| Task boundaries | Explicit (e.g., new class added in Task 2) | Implicit (change in data distribution, new anomaly types without notice) |
| Task identity at test time | Often known or inferable (multi-head models, task ID provided) | Unknown (single model handles all shifts without task ID) |
| Task shifts | Discrete & labeled (new class, new dataset) | Continuous & unlabeled (novelty, concept drift, new attack patterns) |

Challenge II:

IC: Tasks are explicit

- New classes or domains clearly labeled
 - e.g., Task 1: CIFAR-10, Task 2: CIFAR-100

AD: Task changes are often implicit

- Models face distribution shifts & concept drifts without knowledge of "which task" they are in.
- New anomalies appear without labels, warnings, or task boundaries

Example in real-world domains:

- **Cybersecurity:** Evolving network services and user behaviors
- **Finance:** New transaction patterns emerge over time
- **Industrial IoT:** New devices, teardown of existing devices, etc.

Challenge III:

Evolving definition of normal class where normal data from one task may be anomalous in another task (*task-specific characterization of normal class*).

Industrial Predictive Maintenance

- A machine's behavior under **normal load patterns in one season (e.g., winter production)** could be labeled normal.
- The *same operating pattern* in summer (e.g., overheating risk) may indicate an anomaly due to environmental changes.
- Maintenance procedures or upgrades can also **change the expected "normal" operational signature.**

Challenge III:

Evolving definition of normal class where normal data from one task may be anomalous in another task (*task-specific characterization of normal class*).

Medical Healthcare

Patient population shifts (e.g., age, disease prevalence):

- A blood pressure reading considered **normal in a 25-year-old** may be **anomalous in an elderly patient**.
- A **high heart rate** while **running** is considered **normal**, and during **resting/sleeping** it would be **anomaly**
- This is a classic challenge when deploying machine learning systems across hospitals or demographics.

Challenge IV:

Models challenged to **incorporate new behaviors** of the normal class, while **retaining knowledge** of previously observed behaviors and being **exposed to new types of anomalies** (*semi-supervised stability-plasticity*).

Credit Card Fraud Detection

Consumer habits evolve:

- Buying patterns shift (e.g., online subscriptions rise, international travel declines or rises).
- The model must:
 - Update its concept of **normal spending behavior**;
 - Maintain ability to **detect previously known fraud patterns**;
 - Detect *new fraud techniques* (e.g., synthetic IDs, virtual cards).

Challenge IV:

Models challenged to **incorporate new behaviors** of the normal class, while **retaining knowledge** of previously observed behaviors and being **exposed to new types of anomalies** (*semi-supervised stability-plasticity*).

Cybersecurity

CICIDS-2017 (Cybersecurity):

- Data provides **different attack types** and **shifts in normal traffic** (e.g., new services, changed traffic patterns).
- Shows the need to generalize to new behaviors while remembering prior normal/attack patterns.

Takeaways

- **Anomaly detection is a different problem than image classification**
- **Existence of unique challenges:**
 - Limited availability of anomalies compared to normal data
 - Lack of information about task changes and task identities
 - Evolving definition of normal class
 - Semi-supervised stability-plasticity
- **It requires specific continual learning approaches**

Questions?



Outline

1. Motivation for continual anomaly detection
2. Challenges of anomaly detection in continual learning
3. Scenarios, metrics, and strategies
4. pyCLAD: A universal framework for continual lifelong anomaly detection

From Continual Image Classification...

Continual Image Classification Scenario Types

- **Task-incremental:** The model is informed about which task is currently being processed during both the training and the inference stage.
- **Class-incremental:** Requires the model to infer the task on its own and provide the classification decision without explicit information about task identity, but with information on task boundary.
- **Task-agnostic:** Does not assume the availability of task boundaries and labels. Requires techniques such as lifelong change point detection or concept drift detection.
- **Domain Incremental:** It challenges the model by changing the data distribution while keeping the same task that the model needs to solve

Continual Image Classification Scenario Types

- **Task-incremental:** The model is informed about which task is currently being processed during both the training and the inference stage.
- **Class-incremental:** Requires the model to infer the task on its own and provide the classification decision without explicit information about task identity, but with information on task boundary.
- **Task-agnostic:** Does not assume the availability of task boundaries and labels. Requires techniques such as lifelong change point detection or concept drift detection.
- **Domain Incremental:** It challenges the model by changing the data distribution while keeping the same task that the model needs to solve

Continual Image Classification Scenario Types

- **Task-incremental:** The model is informed about which task is currently being processed during both the training and the inference stage.
- **Class-incremental:** Requires the model to infer the task on its own and provide the classification decision without explicit information about task identity, but with information on task boundary.
- **Task-agnostic:** Does not assume the availability of task boundaries and labels. Requires techniques such as lifelong change point detection or concept drift detection.
- **Domain Incremental:** It challenges the model by changing the data distribution while keeping the same task that the model needs to solve



Performance Metrics (1)

We want to monitor:

- Performance on **current** experience
- Performance on **past** experiences
- Performance on **future** experiences
- Resource consumption (Memory / CPU / GPU / Disk usage)
- Model size growth (with respect to the first model)
- Execution time

| R | Te_1 | Te_2 | Te_3 |
|--------|-----------|-----------|-----------|
| Tr_1 | $R_{1,1}$ | $R_{1,2}$ | $R_{1,3}$ |
| Tr_2 | $R_{2,1}$ | $R_{2,2}$ | $R_{2,3}$ |
| Tr_3 | $R_{3,1}$ | $R_{3,2}$ | $R_{3,3}$ |

ACC Metric

- After training on all experiences, average accuracy over all the test experiences.

$$\text{Average Accuracy: } \text{ACC} = \frac{1}{T} \sum_{i=1}^T R_{T,i}$$

A Metric

- Average of the accuracy on all experiences at any point in time.

$$A = \frac{\sum_{i=1}^N \sum_{j=1}^i R_{i,j}}{\frac{N(N+1)}{2}}$$

Source: ContinualAI

Performance Metrics (2)

FWT Metric

- Accuracy on experience i after training on last experience Minus
- Accuracy on experience i before training on the first experience (model init)
- Averaged over $i=2, \dots, T$

| R | Te_1 | Te_2 | Te_3 |
|--------|-----------|-----------|-----------|
| Tr_1 | $R_{1,1}$ | $R_{1,2}$ | $R_{1,3}$ |
| Tr_2 | $R_{2,1}$ | $R_{2,2}$ | $R_{2,3}$ |
| Tr_3 | $R_{3,1}$ | $R_{3,2}$ | $R_{3,3}$ |

$$FWT = \frac{1}{T-1} \sum_{i=2}^T R_{i-1,i} - \bar{b}_i$$

BWT Metric

- Accuracy on experience i after training on experience T Minus
- Accuracy on experience i after training on experience i
- Averaged over $i=1, \dots, T-1$

FORGETTING = - BWT

$$BWT = \frac{1}{T-1} \sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$$

From Continual Image Classification
to
Continual Anomaly Detection

Lifelong Continual Learning for Anomaly Detection: New Challenges, Perspectives, and Insights

Publisher: IEEE

Cite This

PDF

Kamil Faber  ; Roberto Corizzo  ; Bartłomiej Sniezynski  ; Nathalie Japkowicz  All Authors

<https://ieeexplore.ieee.org/document/10473036>



Concept

We define a self-consistent behavior of the normal class, alongside the specific anomalies occurring with it, as a **concept**.

A behavior could correspond to a new distribution, change of a performed activity, or a new state of the environment, depending on the specific analytical context considered.

Example 1: Monitoring human activities

Resting

Sleeping

Working

Jogging

Example 2: Intrusion detection in cloud environment

Server 1

Wordpress websites

Server 2

Video content

Server 3

Gaming server

Server 4

LLM Chat

Continual Anomaly Detection Scenario Types

- **Concept-aware:**
 - Known concept identifier and concept boundaries.
- **Concept-incremental:**
 - Unknown concept identifier but known concept boundaries.
- **Concept-agnostic:**
 - Unknown concept identifier and concept boundaries.

<https://github.com/lifelonglab/lifelong-anomaly-detection-scenarios>

Scenario creation algorithm

Input: c – Number of desired concepts

Input: \mathbf{N}, \mathbf{A} – Normal/Anomaly data

Input: ϕ – Concepts creation function for normal data

Input: γ – Concepts creation function for anomalies

Input: λ – Assignment function

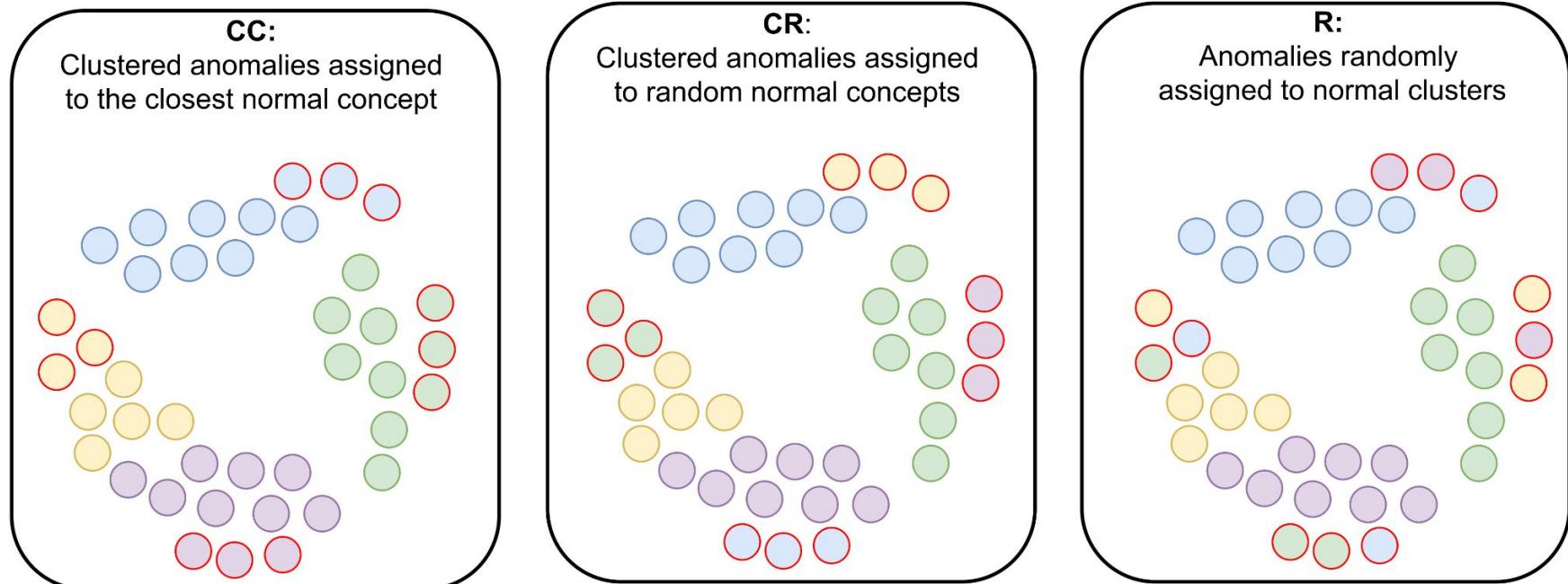
```
1  $C_N \leftarrow \phi(\mathbf{N}, c)$       // Create concepts  $\{C_{N_0}, C_{N_1}, \dots, C_{N_c}\}$ 
2  $C_A \leftarrow \gamma(\mathbf{A}, c)$       // Create concepts  $\{C_{A_0}, C_{A_1}, \dots, C_{A_c}\}$ 
3  $T \leftarrow \emptyset$                   // Result scenario
4 for  $C_{N_i} \in C_N$  do
5    $j \leftarrow \lambda(C_A, C_{N_i})$     // Match anomaly-normal concepts
6    $T \leftarrow T \cup (C_{N_i}, C_{A_j})$  // Add concepts to scenario
7    $C_A \leftarrow C_A - C_{A_j}$        // Remove used anomaly concept
8 end
9 return  $T$ 
```

<https://github.com/lifelonglab/lifelong-anomaly-detection-scenarios>

Scenario creation algorithm



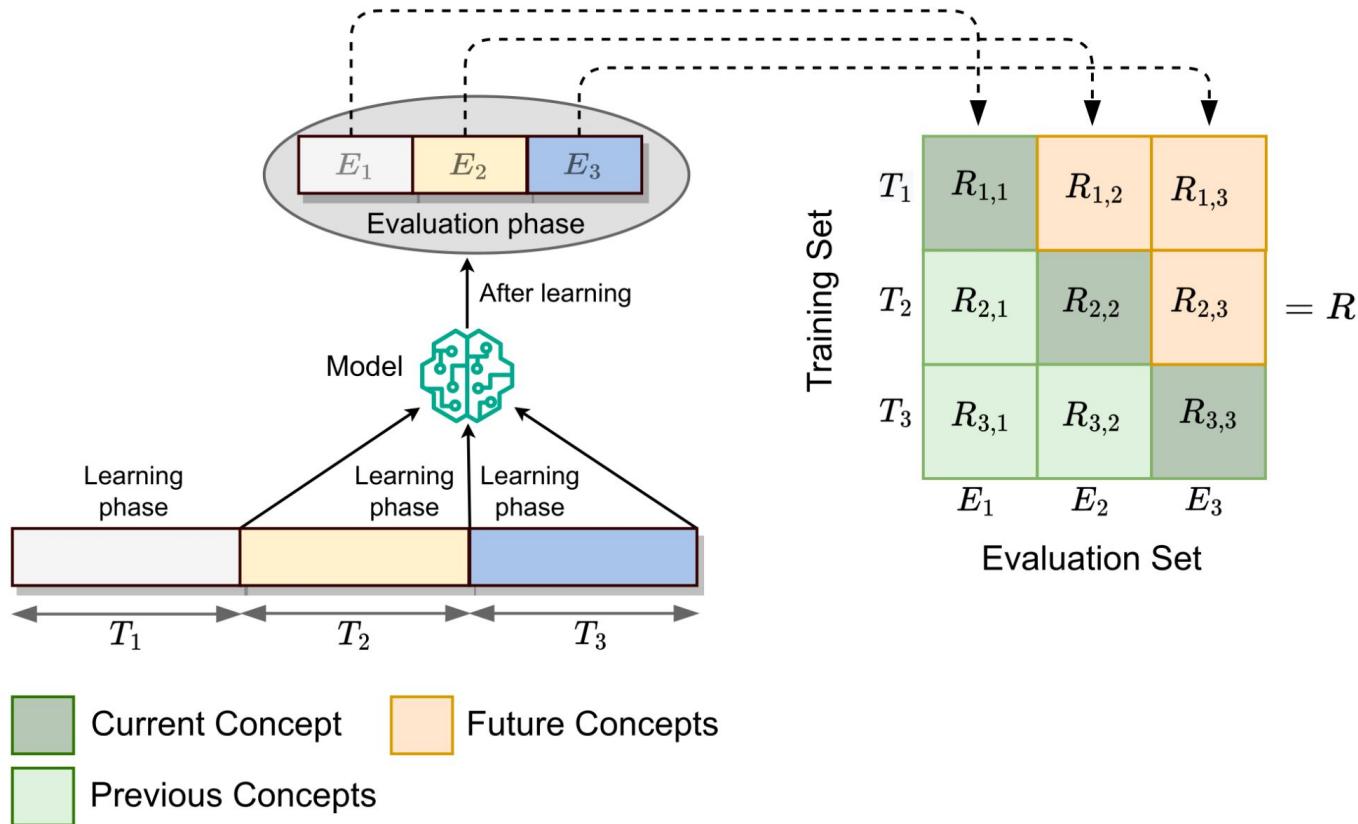
Scenario creation algorithm



Normal data
Anomaly

| Concept 0 | Concept 1 | Concept 2 | Concept 3 |
|---------------|-------------|--------------|---------------|
| Yellow circle | Blue circle | Green circle | Purple circle |
| Red circle | Red circle | Red circle | Red circle |

Evaluation protocol



Metrics

$$\text{Lifelong ROC-AUC} = \frac{\sum_{i \geq j}^N R_{i,j}}{\frac{N(N+1)}{2}}$$

$$BWT = \frac{\sum_{i=2}^N \sum_{j=1}^{i-1} R_{i,j} - R_{j,j}}{\frac{N(N-1)}{2}}$$

$$FWT = \frac{\sum_{i < j}^N R_{i,j}}{\frac{N(N-1)}{2}}$$

Research question

RQ1: Do lifelong scenarios impact the performance of non-lifelong anomaly detection models?

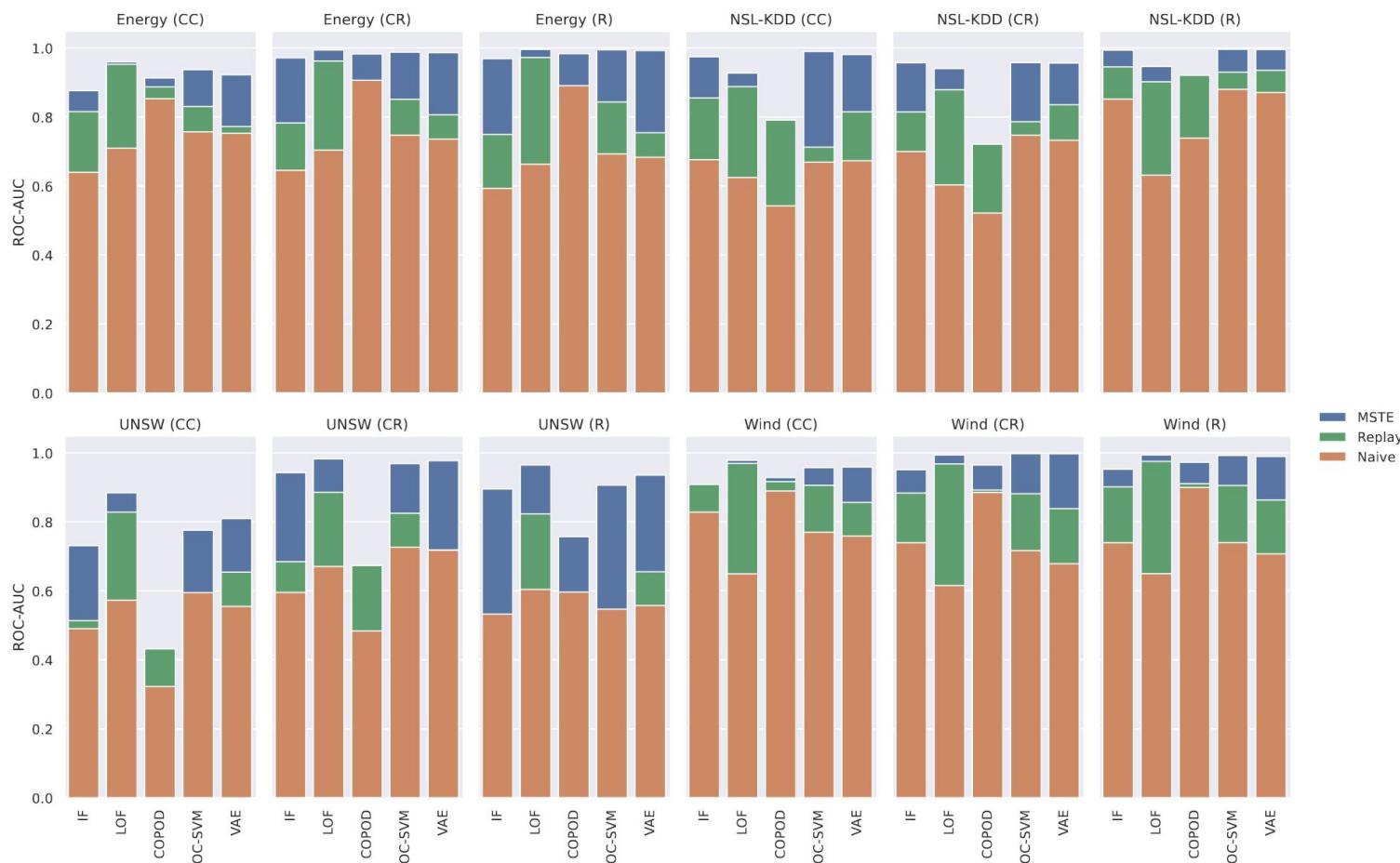
RQ2: Does the adoption of knowledge retention capabilities of lifelong learning provide a valuable improvement in the learning capabilities of existing anomaly detection models in complex lifelong scenarios?



Does lifelong learning make sense in anomaly detection?

Strategies

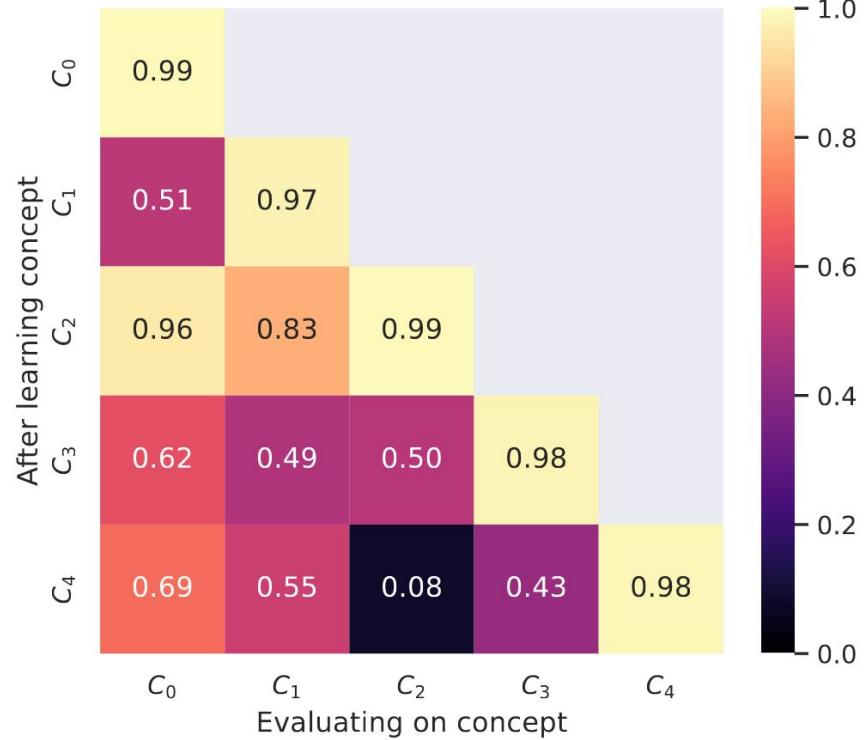
- **Naive:**
 - models are updated as new data becomes available, without any smart lifelong learning strategy to tune adaptation and knowledge retention.
- **Multiple Single-Task Experts (MSTE):**
 - a way to simulate upper-bound model performance in a non-lifelong scenario. In this strategy, a pool or ensemble of models, each of which is an expert for a single concept, is adopted.
- **Replay:**
 - a replay-based method that preserves selected data samples from previous concepts in a memory buffer, which is limited in size by a parameter known as a budget. When the model faces a new task (concept), the replay buffer is updated to include the data from the new concept.



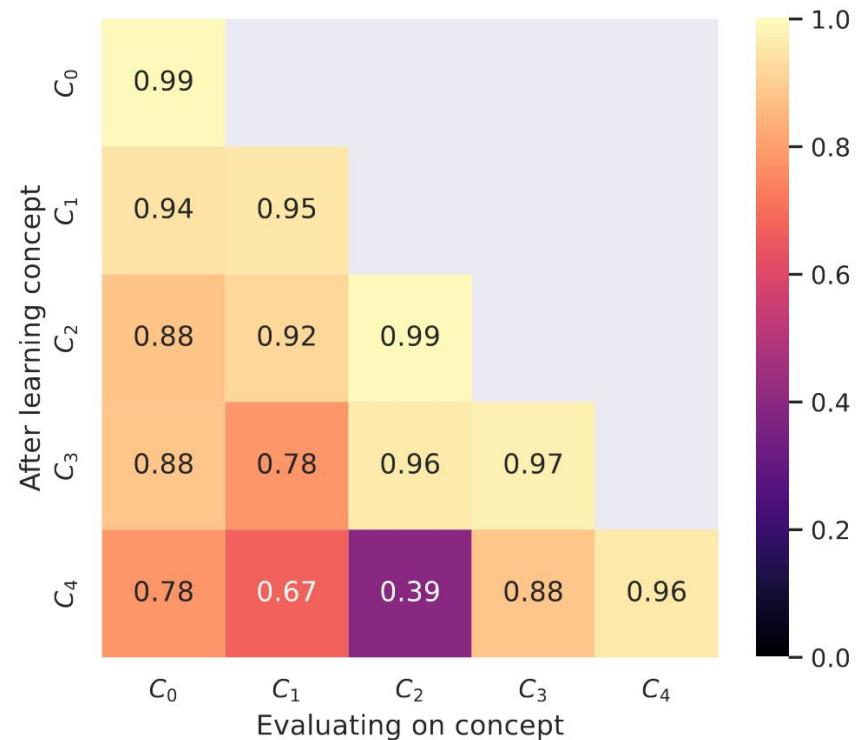
The results illustrating the performance gap between non-lifelong and lifelong strategies in lifelong anomaly detection scenarios.

RQ1: Naive vs MSTE

RQ2: Naive vs Replay



Naive strategy



Replay strategy

Takeaways

- **Performance gap** between non-LL/CL and LL/CL learning strategies
 - LL/CL scenarios are challenging for non-LL/CL anomaly detection methods.
- Strategies such as **Replay** can deal with these challenges.
- **Continual learning is essential for anomaly detection**
 - Real-life complexity to the experimental setting
 - Advantages compared to static and online scenarios

Questions?



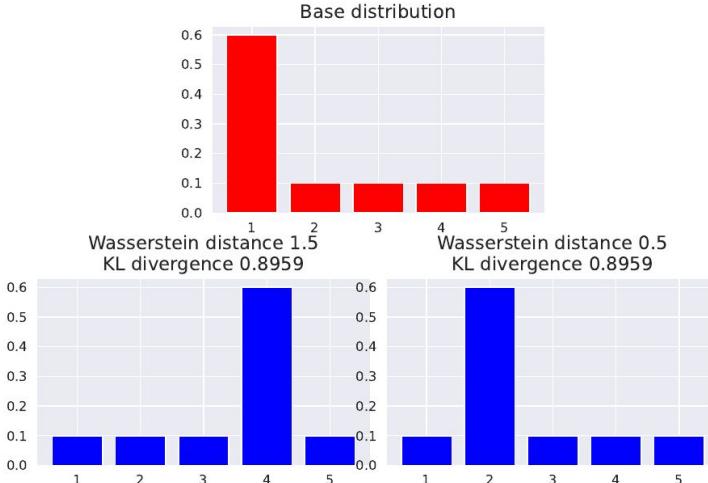
Recent Research and Open Avenues

Task-agnostic Anomaly Detection

WATCH: Wasserstein Change Point Detection for High-Dimensional Time Series Data (*IEEE BigData 2021*)

LIFEWATCH: Lifelong Wasserstein Change Point Detection (*IJCNN 2022*)

- Wasserstein distance



Task-agnostic Anomaly Detection

LIFEWATCH: Lifelong Wasserstein Change Point Detection

- Detecting changes between tasks
- Detecting recurrent changes
- Recognizing which task is currently being processed (also recurring tasks).

Task-agnostic Anomaly Detection

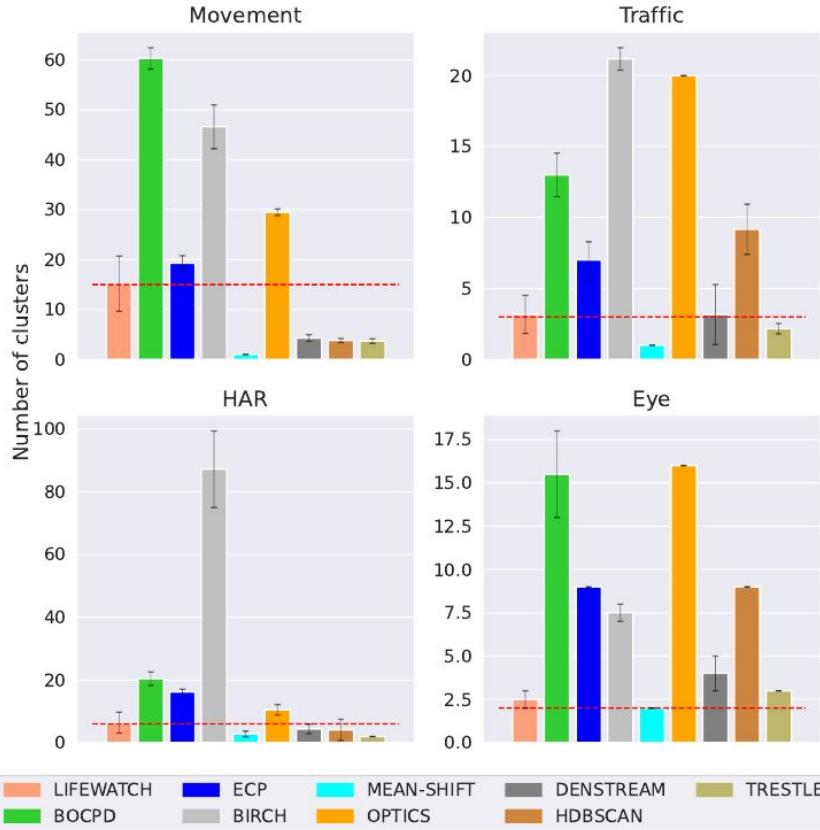
LIFEWATCH: Lifelong Wasserstein Change Point Detection

- Pool of already discovered distributions: P
- Keep track of what is current distribution D_C
- Process data in small mini-batches B_i
- Each distribution D_j has a threshold $E[D_j]$

The threshold helps in determining to which distribution new data belongs to.

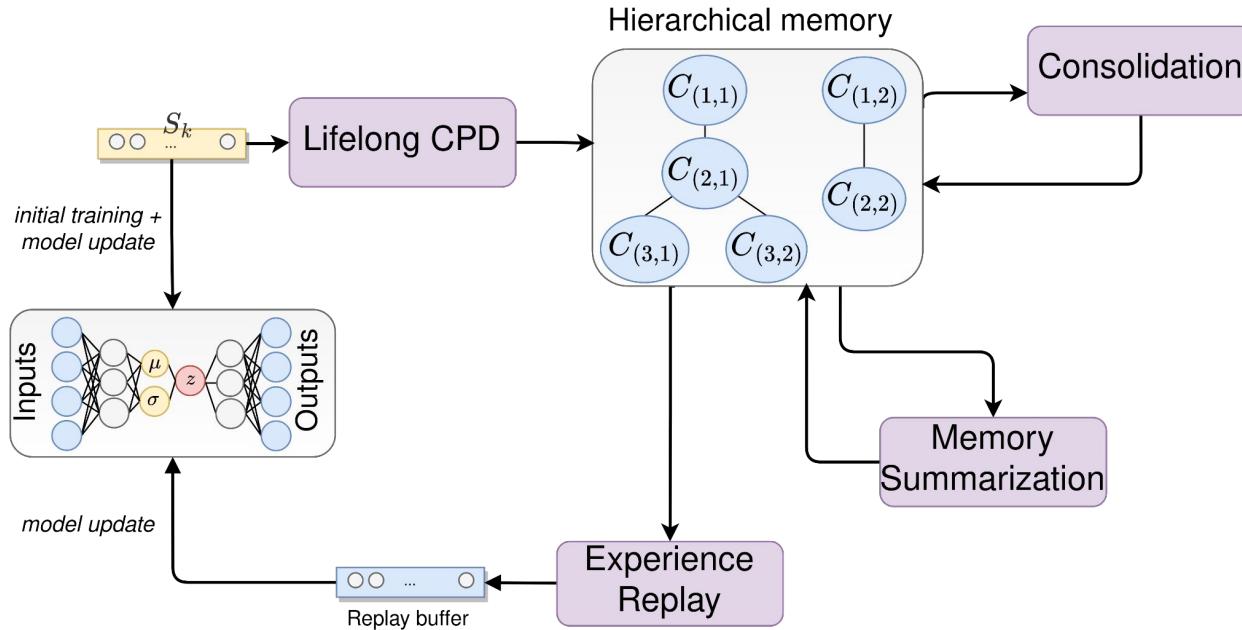
$$E[D_j] = \epsilon \max_{B_i \in D_j} W_A(B_i, D_j).$$

Task-agnostic Anomaly Detection



- Human Activity Recognition (561 features)
- Libras movement (90 features)
- Urban traffic (17 features)
- EEG Measurement with eye open/closed (14 features)

Task-agnostic Anomaly Detection



ELSEVIER

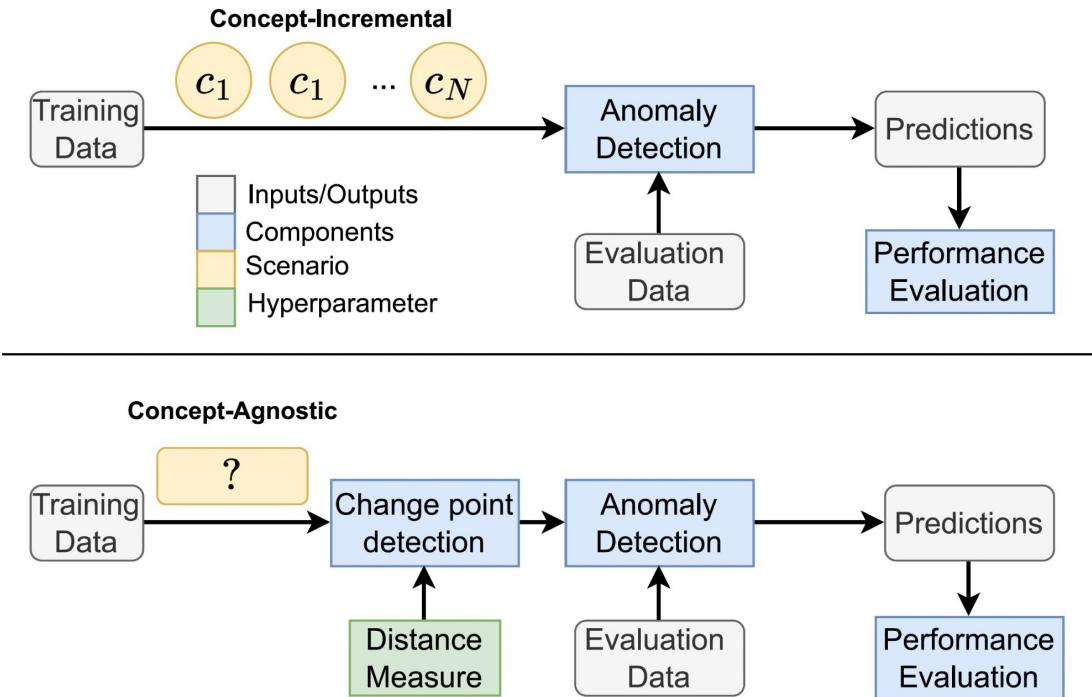
Neural Networks

Volume 165, August 2023, Pages 248-273



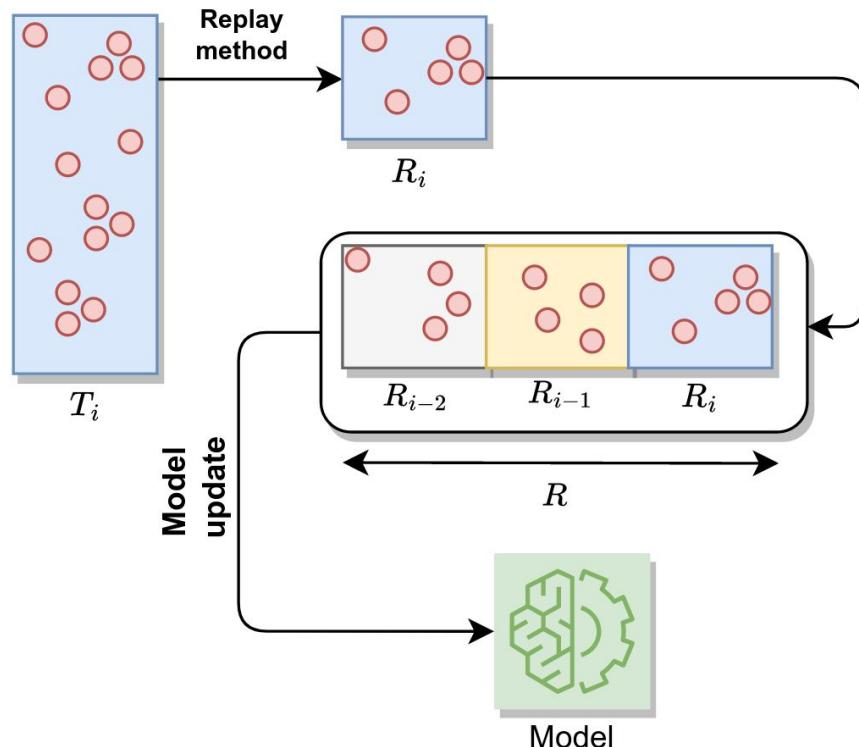
Faber, K., Corizzo, R., Sniezynski, B., & Japkowicz, N. (2023). **VLAD: Task-agnostic vae-based lifelong anomaly detection**. *Neural Networks*, 165, 248-273.

Change Detection for Novelty Detection



Coil, C., Faber, K., Sniezynski, B., & Corizzo, R. (2025). **Distance-based change point detection for novelty detection in concept-agnostic continual anomaly detection.** *Journal of Intelligent Information Systems*, 1-39.

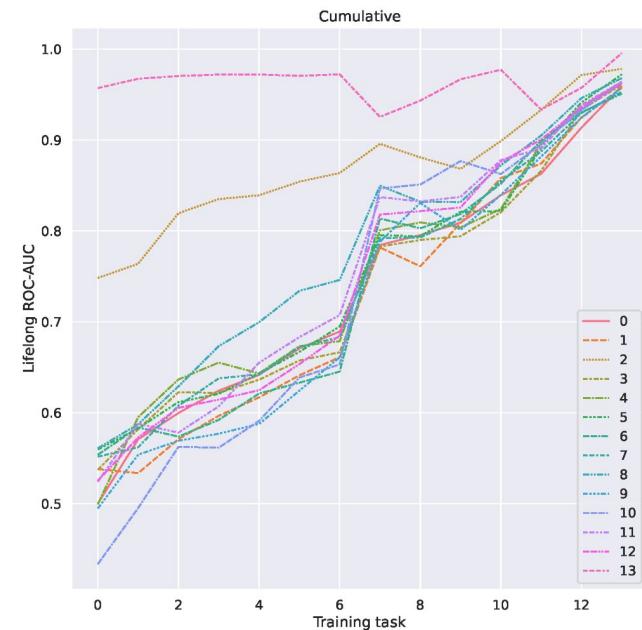
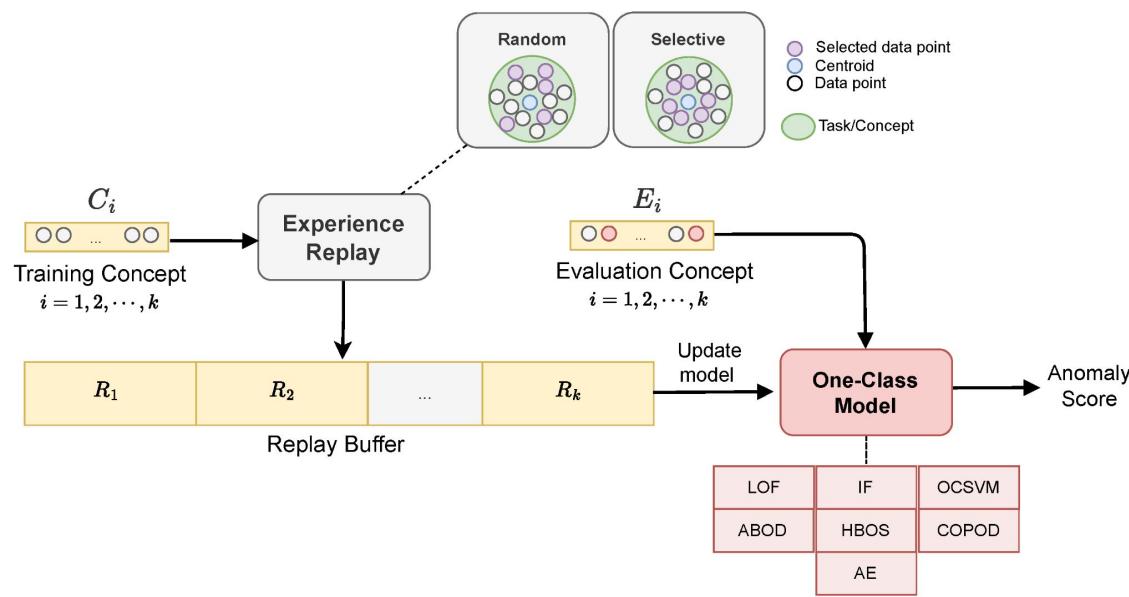
Handle Contamination in Learning Scenarios



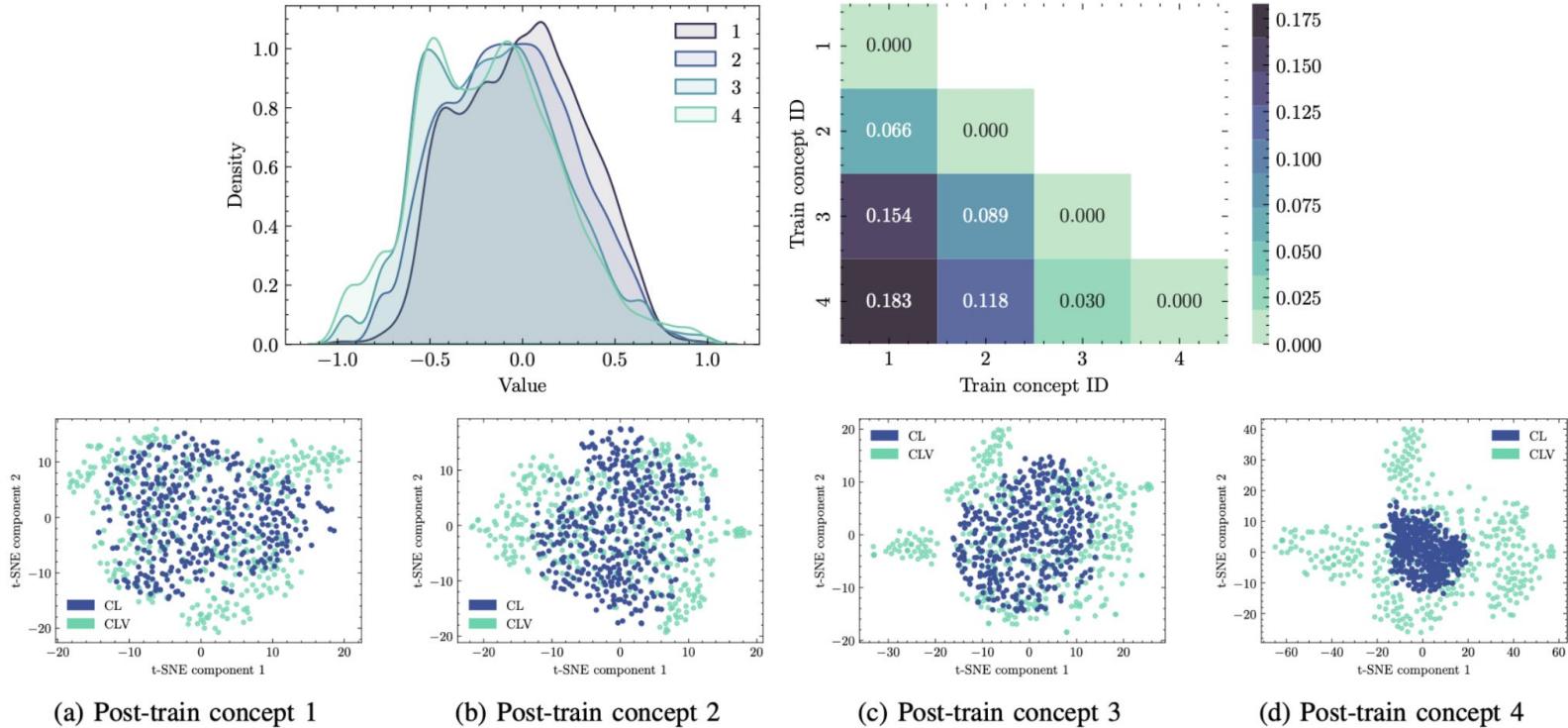
Faber, K., Corizzo, R., Sniezynski, B., & Japkowicz, N. (2022). **Active lifelong anomaly detection with experience replay**. In 2022 IEEE 9th international conference on data science and advanced analytics (DSAA) (pp. 1-10). IEEE.

Different Applications and Domains

Malware Detection



Continual Learning in Cloud Computing



(a) Post-train concept 1

(b) Post-train concept 2

(c) Post-train concept 3

(d) Post-train concept 4

Rethinking Continual Strategies for Image Classification

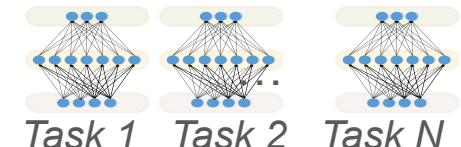
- **Replay-based**
 - Store experiences from past episodes and replay them while training with new tasks
 - **GEM, A-GEM, GDumb**
- **Regularization-based**
 - Put constraints on the loss function to prevent losing knowledge of already learned patterns
 - **SI, LwF, EWC, LODE**
- **Architectural-based**
 - Focus on the topology of the neural model trying to alter it or leverage the available capacity to prevent the model from forgetting
 - **CWRStar, PackNet, WSN, Ada-Q-PackNet, DyTox**
- **Hybrid**
 - Provide a mixture of regularization, memory-based, and architectural approaches.
 - **NPCL, QDI, Pro-KT, SGP**

Rethinking Continual Strategies for Image Classification

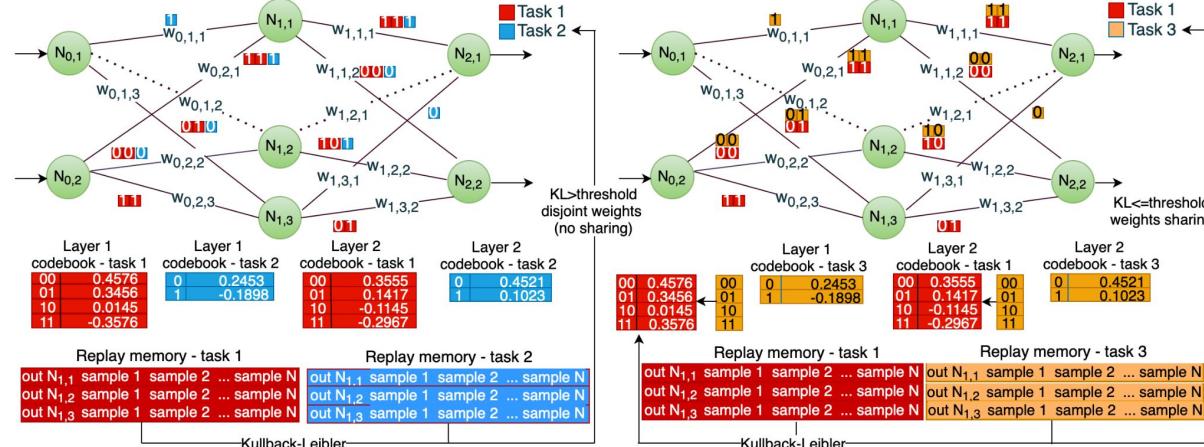
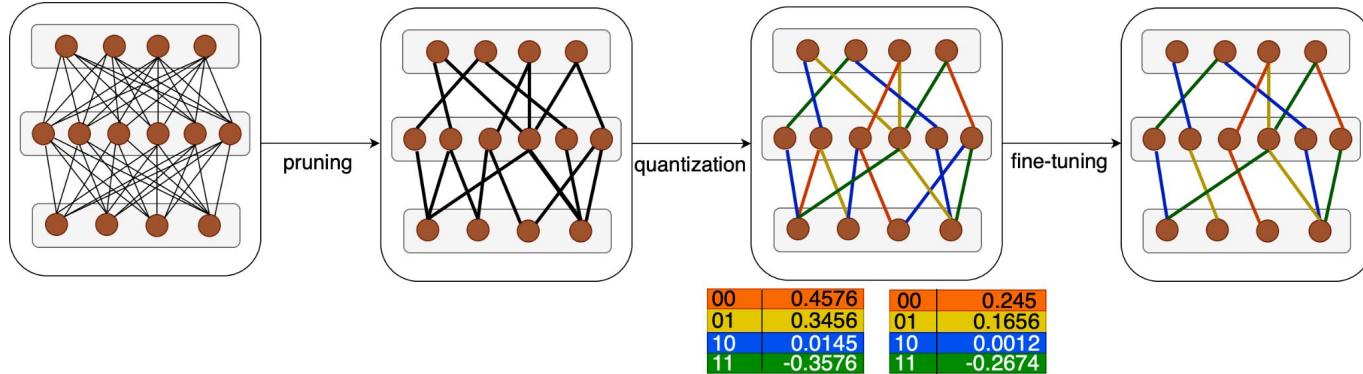
- **Replay-based**
 - Store experiences from past episodes and replay them while training with new tasks
 - **GEM, A-GEM, GDumb**
- **Regularization-based**
 - Put constraints on the loss function to prevent losing knowledge of already learned patterns
 - **SI, LwF, EWC, LODE**
- **Architectural-based**
 - Focus on the topology of the neural model trying to alter it or leverage the available capacity to prevent the model from forgetting
 - **CWRStar, PackNet, WSN, Ada-Q-PackNet, DyTox**
- **Hybrid**
 - Provide a mixture of regularization, memory-based, and architectural approaches.
 - **NPCL, QDI, Pro-KT, SGP**

Rethinking Continual Strategies for Image Classification

- **Replay-based**
 - Store experiences from past episodes and replay them while training with new tasks
 - **GEM, A-GEM, GDumb**
- **Regularization-based**
 - Put constraints on the loss function to prevent losing knowledge of already learned patterns
 - **SI, LwF, EWC, LODE**
- **Architectural-based**
 - Focus on the topology of the neural model trying to alter it or leverage the available capacity to prevent the model from forgetting
 - **CWRStar, PackNet, WSN, Ada-Q-PackNet, DyTox**
- **Hybrid**
 - Provide a mixture of regularization, memory-based, and architectural approaches.
 - **NPCL, QDI, Pro-KT, SGP**



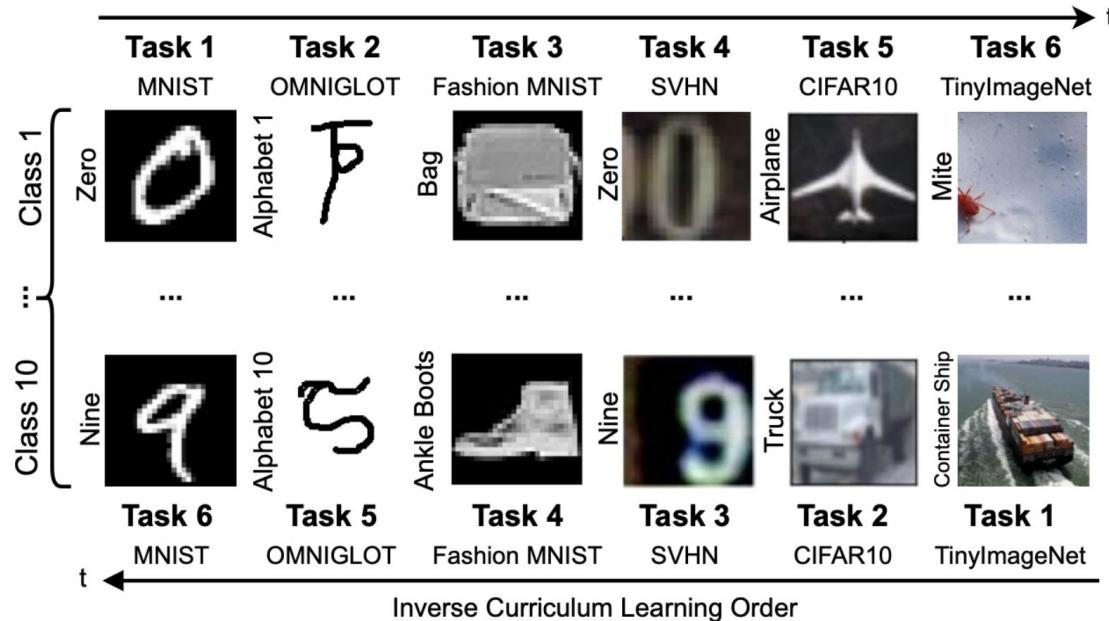
Hybrid Strategies with Compression



Pietron, M., Faber, K., Żurek, D., & Corizzo, R. (2025). **TinySubNets: An efficient and low capacity continual learning strategy.** In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 39, No. 19, pp. 19913-19920).

Exploiting structure in tasks scenarios (*curriculum learning*)

Order matters!



Faber, K., Zurek, D., Pietron, M., Japkowicz, N., Vergari, A., & Corizzo, R. (2024). **From MNIST to ImageNet and back: benchmarking continual curriculum learning.** *Machine Learning*, 113(10), 8137-8164.

Outline

1. Motivation for continual anomaly detection
2. Challenges of anomaly detection in continual learning
3. Scenarios, metrics, and strategies
4. **pyCLAD: A universal framework for continual lifelong anomaly detection**

pyCLAD

- A unified framework for continual anomaly detection.
- Main goal is to foster successful scientific development in continual anomaly detection by providing robust implementations of common functionalities.

How do I install pyCLAD?

- pyCLAD is available as a [Python package on PyPI](#).
- It can be installed using tools such as pip and conda.

Conda

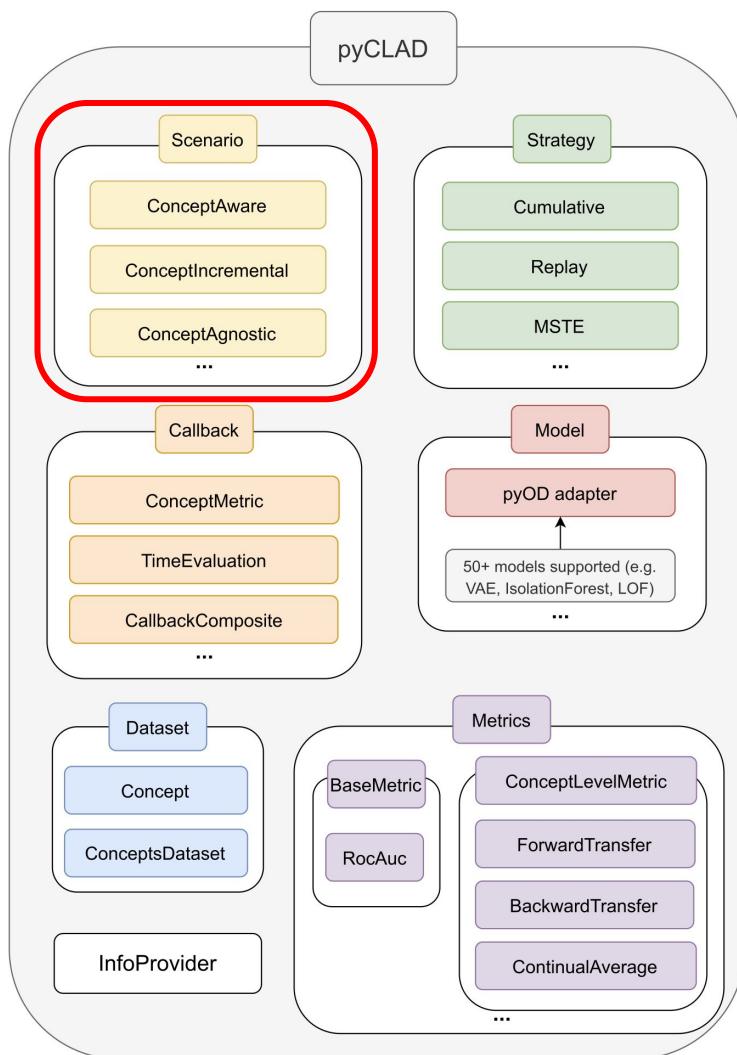
```
conda install -c conda-forge pyclad
```

Pip

```
pip install pyclad
```

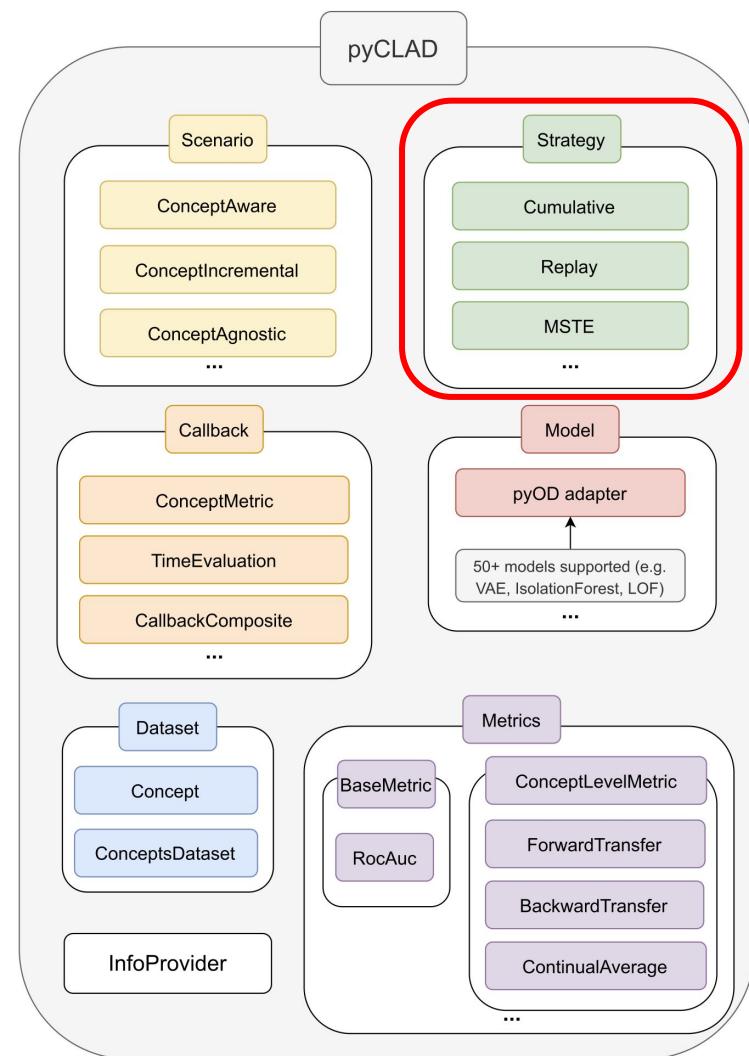
Core concepts

- **Scenario:** It defines the data stream so that it reflects:
 - **Different real-life conditions**
 - **Challenges** faced by the continual strategy



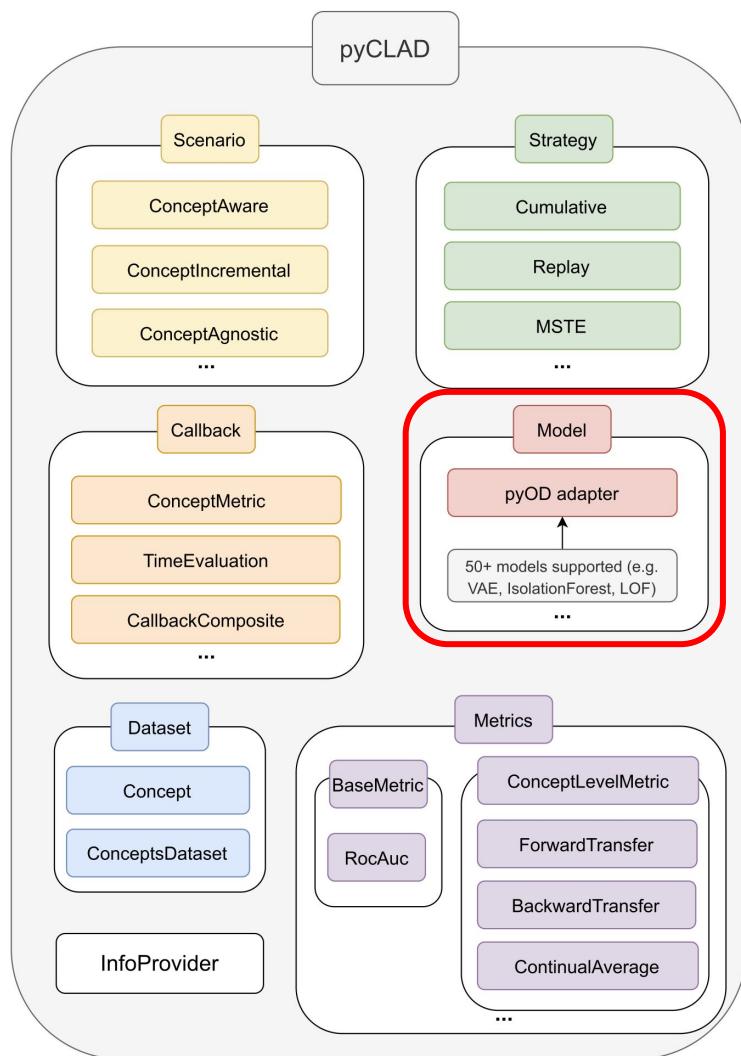
Core concepts

- **Strategy**: A way to manage model updates.
- Responsible for **how, when, and with which data** models should be updated.
- Its aim is to introduce **knowledge retention** while keeping the **ability to adapt**.



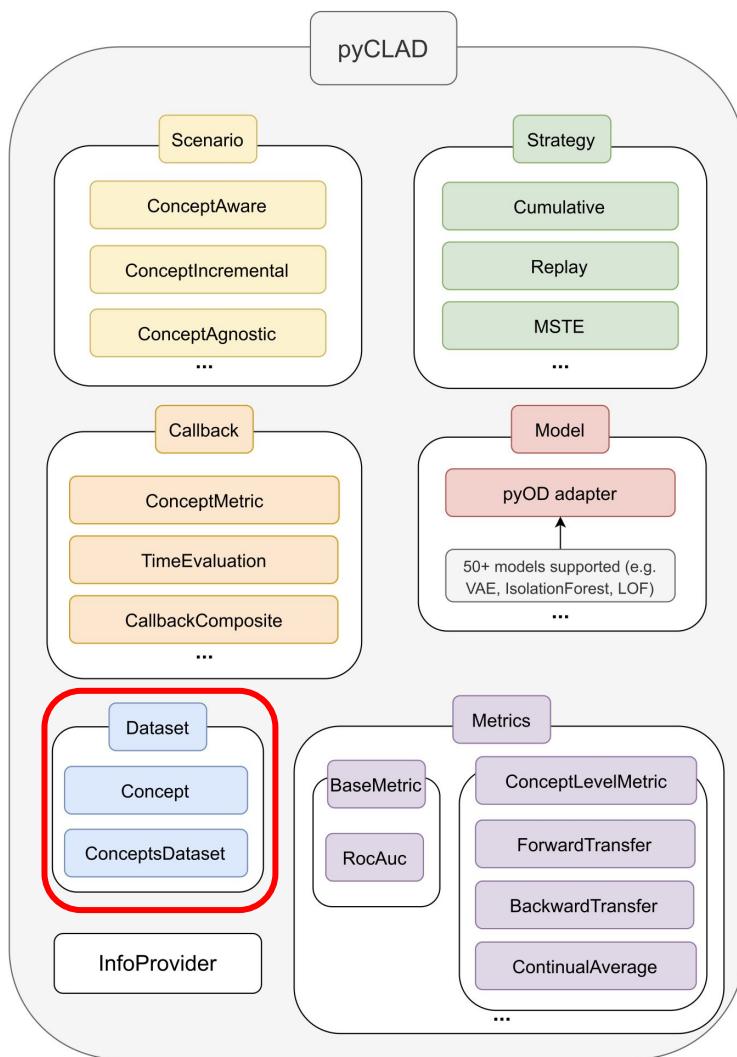
Core concepts

- **Model:** A machine learning model used for anomaly detection.
- Models are often leveraged by continual strategies that add additional layer of managing model's updates.



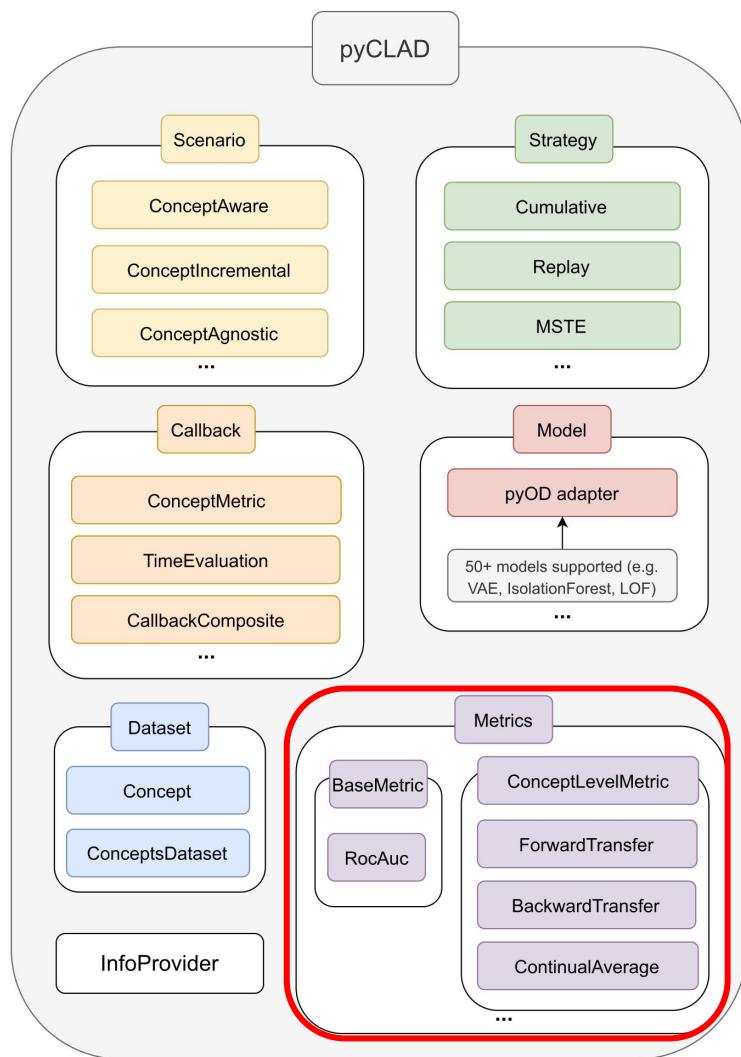
Core concepts

- **Dataset**: A collection of data used for training and evaluation of the model



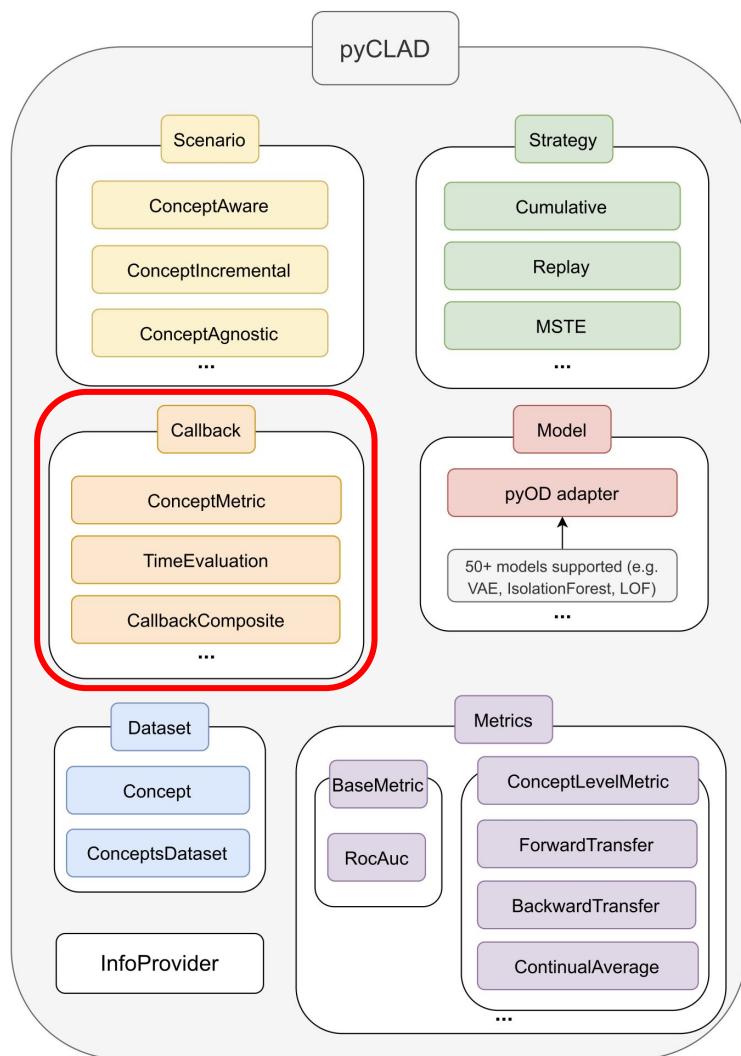
Core concepts

- **Metrics**: A way to evaluate the performance of the model



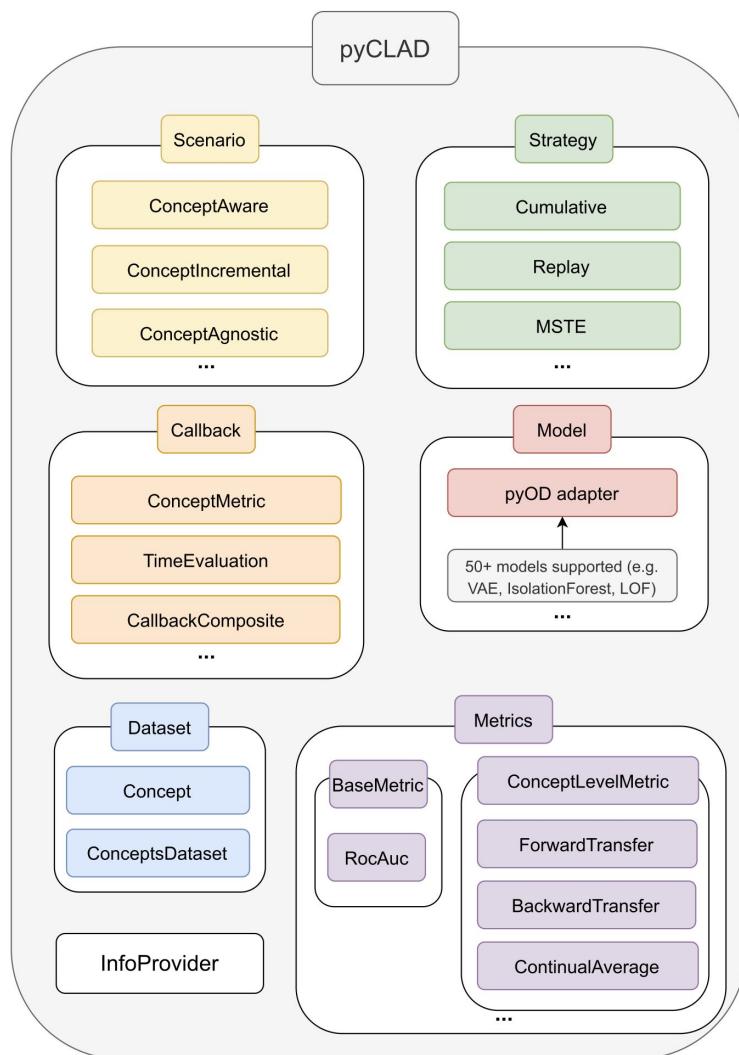
Core concepts

- **Callbacks**: A function that is called at specific points during the scenario.
- Useful for monitoring the process, calculating metrics, and more.



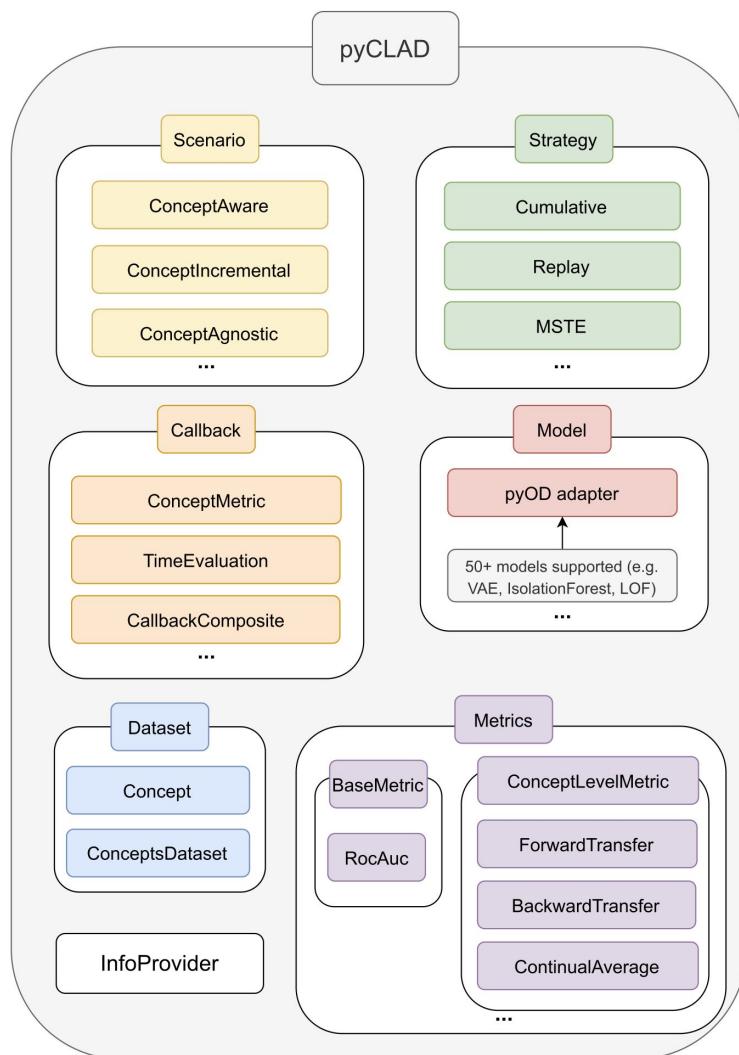
Running an experiment

- Selecting a **Scenario** based on a **Dataset**
- Choosing desired **Strategy**, **Models**, and an evaluation scheme through **Metrics** and **Callbacks**.



Running an experiment

- A user can leverage any dataset by dividing it into multiple **Concepts** and creating **ConceptsDataset**.
- A large variety of anomaly detection models is supported through adapters for the pyOD library



Preparing a Dataset

```
concept1_train = Concept("concept1", data=np.random.rand(100, 10))
concept1_test = Concept("concept1", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))

concept2_train = Concept("concept2", data=np.random.rand(100, 10))
concept2_test = Concept("concept2", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))

concept3_train = Concept("concept3", data=np.random.rand(100, 10))
concept3_test = Concept("concept3", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))
```

Preparing a Dataset

```
concept1_train = Concept("concept1", data=np.random.rand(100, 10))
concept1_test = Concept("concept1", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))

concept2_train = Concept("concept2", data=np.random.rand(100, 10))
concept2_test = Concept("concept2", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))

concept3_train = Concept("concept3", data=np.random.rand(100, 10))
concept3_test = Concept("concept3", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))
```

Preparing a Dataset

```
concept1_train = Concept("concept1", data=np.random.rand(100, 10))
concept1_test = Concept("concept1", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))

concept2_train = Concept("concept2", data=np.random.rand(100, 10))
concept2_test = Concept("concept2", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))

concept3_train = Concept("concept3", data=np.random.rand(100, 10))
concept3_test = Concept("concept3", data=np.random.rand(100, 10),
labels=np.random.randint(0, 2, 100))
```

Preparing a Dataset

```
# Build a dataset based on the previously created concepts
dataset = ConceptsDataset(
    name="GeneratedDataset",
    train_concepts=[concept1_train, concept2_train, concept3_train],
    test_concepts=[concept1_test, concept2_test, concept3_test],
)
```

Defining model

```
# Define model, strategy, and callbacks
model = OneClassSVMAdapter()
strategy = CumulativeStrategy(model)

time_callback = TimeEvaluationCallback()
metric_callback = ConceptMetricCallback(
    base_metric=RocAuc(), metrics=[ContinualAverage(),
BackwardTransfer(), ForwardTransfer()])
)
```

Selecting strategy

```
# Define model, strategy, and callbacks
model = OneClassSVMAdapter()
strategy = CumulativeStrategy(model)

time_callback = TimeEvaluationCallback()
metric_callback = ConceptMetricCallback(
    base_metric=RocAuc(), metrics=[ContinualAverage(),
BackwardTransfer(), ForwardTransfer()])
)
```

Defining callbacks

```
# Define model, strategy, and callbacks
model = OneClassSVMAdapter()
strategy = CumulativeStrategy(model)

time_callback = TimeEvaluationCallback()
metric_callback = ConceptMetricCallback(
    base_metric=RocAuc(), metrics=[ContinualAverage(),
BackwardTransfer(), ForwardTransfer()])
)
```

Creating & executing scenario

```
# Execute the concept agnostic scenario
scenario = ConceptAgnosticScenario(dataset=dataset, strategy=strategy,
callbacks=[metric_callback, time_callback])
scenario.run()

# Save the results
output_writer = JsonOutputWriter(pathlib.Path("output.json"))
output_writer.write([model, dataset, strategy, metric_callback,
time_callback])
```

Saving results

```
# Execute the concept agnostic scenario
scenario = ConceptAgnosticScenario(dataset=dataset, strategy=strategy,
callbacks=[metric_callback, time_callback])
scenario.run()

# Save the results
output_writer = JsonOutputWriter(pathlib.Path("output.json"))
output_writer.write([model, dataset, strategy, metric_callback,
time_callback])
```

Output file

```
{  
    "model": {  
        "name": "OneClassSVM",  
        "cache_size": 200,  
        "coef0": 0.0,  
        "contamination": 0.1,  
        "degree": 3,  
        "gamma": "auto",  
        "kernel": "rbf",  
        "max_iter": -1,  
        "nu": 0.5,  
        "shrinking": true,  
        "tol": 0.001,  
        "verbose": false  
    },  
    "dataset": {  
        "name":  
        "GeneratedDataset",  
        "tran_concepts_no": 3,  
        "test_concepts_no": 3  
    },  
    "strategy": {  
        "name": "Cumulative",  
        "model": "OneClassSVM",  
        "buffer_size": 300  
    },  
}
```

Output file

```
"concept_metric_callback_ROC-AUC": {      "metric_matrix": {
    "base_metric_name": "ROC-AUC",
    "metrics": {
        "ContinualAverage": 0.50746,
        "BackwardTransfer": 0.01811,
        "ForwardTransfer": 0.50441
    },
    "concepts_order": [
        "concept1",
        "concept2",
        "concept3"
    ],
    "metric_matrix": {
        "concept1": {
            "concept1": 0.46698,
            "concept2": 0.50805,
            "concept3": 0.49299
        },
        "concept2": {
            "concept1": 0.48872,
            "concept2": 0.52818,
            "concept3": 0.51220
        },
        "concept3": {
            "concept1": 0.52133,
            "concept2": 0.52818,
            "concept3": 0.51140
        }
    }
},
```

Output file

```
"time_evaluation_callback": {
    "time_by_concept": {
        "concept1": {
            "train_time": 0.00266,
            "eval_time": 0.00505
        },
        "concept2": {
            "train_time": 0.00146,
            "eval_time": 0.00409
        },
        "concept3": {
            "train_time": 0.00251,
            "eval_time": 0.00424
        }
    },
    "train_time_total": 0.00663,
    "eval_time_total": 0.01340
}
```

Extensibility & Implementations

Model class

```
class Model(InfoProvider):  
    @abstractmethod  
    def fit(self, data: np.ndarray): ...  
  
    @abstractmethod  
    def predict(self, data: np.ndarray) -> (np.ndarray, np.ndarray):  
        """  
        :param data:  
        :return: (predicted labels (0 for normal class, 1 for anomaly),  
                 anomaly scores (the higher the more anomalous))  
        """  
  
    ...  
  
    @abc.abstractmethod  
    def name(self) -> str: ...  
  
    def info(self) -> Dict[str, Any]:  
        return {"model": {"name": self.name(), **self.additional_info()}}  
  
    def additional_info(self):  
        return {}
```

```
class InfoProvider(abc.ABC):  
  
    @abc.abstractmethod  
    def info(self) -> Dict[str, Any]:  
        ...
```

Models

- PyOD
- PyTorch
- Anything :)

```
model = PyODAdapter(  
    VAE(  
        encoder_neuron_list=[32, 24, 16],  
        decoder_neuron_list=[16, 24, 32],  
        latent_dim=8,  
        epoch_num=20,  
        preprocessing=False,  
    ),  
    model_name="VAE",  
)
```

Implementing an AutoEncoder model

```
class Autoencoder(Model):
    def __init__(self, encoder: nn.Module, decoder: nn.Module, lr: float = 1e-2, threshold: float = 0.5, epochs: int = 20):
        self.module = AutoencoderModule(encoder, decoder, lr)
        self.threshold = threshold
        self.epochs = epochs

    def fit(self, data: np.ndarray):
        dataset = TensorDataset(torch.Tensor(data))
        dataloader = torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=True)
        trainer = pl.Trainer(max_epochs=self.epochs)
        trainer.fit(self.module, dataloader)

    def predict(self, data: np.ndarray) -> (np.ndarray, np.ndarray):
        x_hat = self.module(torch.Tensor(data)).detach()
        rec_error = ((data - x_hat.numpy()) ** 2).mean(axis=1)

        binary_predictions = (rec_error > self.threshold).astype(int)
        return binary_predictions, rec_error

    def name(self) -> str:
        return "Autoencoder"

    def additional_info(self):
        return {
            "threshold": self.threshold,
            "encoder": str(self.module.encoder),
            "decoder": str(self.module.decoder),
            "lr": self.module.lr,
            "epochs": self.epochs,
        }
```

Implementing an AutoEncoder model

```
class Autoencoder(Model):
    def __init__(self, encoder: nn.Module, decoder: nn.Module, lr: float = 1e-2, threshold: float = 0.5, epochs: int = 20):
        self.module = AutoencoderModule(encoder, decoder, lr)
        self.threshold = threshold
        self.epochs = epochs

    def fit(self, data: np.ndarray):
        dataset = TensorDataset(torch.Tensor(data))
        dataloader = torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=True)
        trainer = pl.Trainer(max_epochs=self.epochs)
        trainer.fit(self.module, dataloader)

    def predict(self, data: np.ndarray) -> (np.ndarray, np.ndarray):
        x_hat = self.module(torch.Tensor(data)).detach()
        rec_error = ((data - x_hat.numpy()) ** 2).mean(axis=1)

        binary_predictions = (rec_error > self.threshold).astype(int)
        return binary_predictions, rec_error

    def name(self) -> str:
        return "Autoencoder"

    def additional_info(self):
        return {
            "threshold": self.threshold,
            "encoder": str(self.module.encoder),
            "decoder": str(self.module.decoder),
            "lr": self.module.lr,
            "epochs": self.epochs,
        }
```

Implementing an AutoEncoder model

```
class Autoencoder(Model):
    def __init__(self, encoder: nn.Module, decoder: nn.Module, lr: float = 1e-2, threshold: float = 0.5, epochs: int = 20):
        self.module = AutoencoderModule(encoder, decoder, lr)
        self.threshold = threshold
        self.epochs = epochs

    def fit(self, data: np.ndarray):
        dataset = TensorDataset(torch.Tensor(data))
        dataloader = torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=True)
        trainer = pl.Trainer(max_epochs=self.epochs)
        trainer.fit(self.module, dataloader)

    def predict(self, data: np.ndarray) -> (np.ndarray, np.ndarray):
        x_hat = self.module(torch.Tensor(data)).detach()
        rec_error = ((data - x_hat.numpy()) ** 2).mean(axis=1)

        binary_predictions = (rec_error > self.threshold).astype(int)
        return binary_predictions, rec_error

    def name(self) -> str:
        return "Autoencoder"

    def additional_info(self):
        return {
            "threshold": self.threshold,
            "encoder": str(self.module.encoder),
            "decoder": str(self.module.decoder),
            "lr": self.module.lr,
            "epochs": self.epochs,
        }
```

Implementing an AutoEncoder model

```
class Autoencoder(Model):
    def __init__(self, encoder: nn.Module, decoder: nn.Module, lr: float = 1e-2, threshold: float = 0.5, epochs: int = 20):
        self.module = AutoencoderModule(encoder, decoder, lr)
        self.threshold = threshold
        self.epochs = epochs

    def fit(self, data: np.ndarray):
        dataset = TensorDataset(torch.Tensor(data))
        dataloader = torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=True)
        trainer = pl.Trainer(max_epochs=self.epochs)
        trainer.fit(self.module, dataloader)

    def predict(self, data: np.ndarray) -> (np.ndarray, np.ndarray):
        x_hat = self.module(torch.Tensor(data)).detach()
        rec_error = ((data - x_hat.numpy()) ** 2).mean(axis=1)

        binary_predictions = (rec_error > self.threshold).astype(int)
        return binary_predictions, rec_error

    def name(self) -> str:
        return "Autoencoder"

    def additional_info(self):
        return {
            "threshold": self.threshold,
            "encoder": str(self.module.encoder),
            "decoder": str(self.module.decoder),
            "lr": self.module.lr,
            "epochs": self.epochs,
        }
```

Implementing an AutoEncoder model

```
class AutoencoderModule(pl.LightningModule):
    def __init__(self, encoder: nn.Module, decoder: nn.Module, lr: float = 1e-2):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.lr = lr

        self.save_hyperparameters()
        self.train_loss = nn.MSELoss()
        self.val_loss = nn.MSELoss()

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

    def training_step(self, batch, batch_idx):
        x = batch[0]
        x_hat = self(x)
        loss = self.train_loss(x_hat, x)
        self.log("train_loss", loss)
        return loss

    def validation_step(self, batch, batch_idx):
        x = batch[0]
        x_hat = self(x)
        loss = self.val_loss(x_hat, x)
        self.log("val_loss", loss)

    def configure_optimizers(self) -> OptimizerLRScheduler:
        return torch.optim.Adam(self.parameters(), lr=self.lr)
```

Callbacks: Abstract Implementation

```
class Callback(abc.ABC):
    def before_scenario(self, *args, **kwargs):
        pass

    def after_scenario(self, *args, **kwargs):
        pass

    def before_training(self, *args, **kwargs):
        pass

    def after_training(self, *args, **kwargs):
        pass

    def before_evaluation(self, *args, **kwargs):
        pass

    def after_evaluation(self, *args, **kwargs):
        pass

    def before_concept_processing(self, *args, **kwargs):
        pass

    def after_concept_processing(self, *args, **kwargs):
        pass
```

Callbacks: Concrete Implementation Example

```
class TimeEvaluationCallback(Callback, InfoProvider):
    def __init__(self):
        self._time_by_concept = defaultdict(lambda: dict({"train_time": 0, "eval_time": 0}))
        self._train_start = 0
        self._eval_start = 0
        self._train_time_total = 0
        self._eval_time_total = 0

    def before_training(self, *args, **kwargs):
        self._train_start = time.time()

    def after_training(self, learned_concept: Concept):
        train_time = time.time() - self._train_start
        self._time_by_concept[learned_concept.name][ "train_time" ] = train_time
        self._train_time_total = self._train_time_total + train_time

    def before_evaluation(self, *args, **kwargs):
        self._eval_start = time.time()

    def after_evaluation(self, evaluated_concept: Concept, *args, **kwargs):
        eval_time = time.time() - self._eval_start
        self._eval_time_total = self._eval_time_total + eval_time
        self._time_by_concept[evaluated_concept.name][ "eval_time" ] += eval_time

    def info(self) -> Dict[str, Any]:
        return {
            "time_evaluation_callback": {
                "time_by_concept": self._time_by_concept,
                "train_time_total": self._train_time_total,
                "eval_time_total": self._eval_time_total,
            }
        }
```

Callbacks: Concrete Implementation Example

```
class TimeEvaluationCallback(Callback, InfoProvider):
    def __init__(self):
        self._time_by_concept = defaultdict(lambda: dict({"train_time": 0, "eval_time": 0}))
        self._train_start = 0
        self._eval_start = 0
        self._train_time_total = 0
        self._eval_time_total = 0

    def before_training(self, *args, **kwargs):
        self._train_start = time.time()

    def after_training(self, learned_concept: Concept):
        train_time = time.time() - self._train_start
        self._time_by_concept[learned_concept.name]["train_time"] = train_time
        self._train_time_total = self._train_time_total + train_time

    def before_evaluation(self, *args, **kwargs):
        self._eval_start = time.time()

    def after_evaluation(self, evaluated_concept: Concept, *args, **kwargs):
        eval_time = time.time() - self._eval_start
        self._eval_time_total = self._eval_time_total + eval_time
        self._time_by_concept[evaluated_concept.name]["eval_time"] += eval_time

    def info(self) -> Dict[str, Any]:
        return {
            "time_evaluation_callback": {
                "time_by_concept": self._time_by_concept,
                "train_time_total": self._train_time_total,
                "eval_time_total": self._eval_time_total,
            }
        }
```

Callbacks: Concrete Implementation Example

```
class TimeEvaluationCallback(Callback, InfoProvider):
    def __init__(self):
        self._time_by_concept = defaultdict(lambda: dict({"train_time": 0, "eval_time": 0}))
        self._train_start = 0
        self._eval_start = 0
        self._train_time_total = 0
        self._eval_time_total = 0

    def before_training(self, *args, **kwargs):
        self._train_start = time.time()

    def after_training(self, learned_concept: Concept):
        train_time = time.time() - self._train_start
        self._time_by_concept[learned_concept.name]["train_time"] = train_time
        self._train_time_total = self._train_time_total + train_time

    def before_evaluation(self, *args, **kwargs):
        self._eval_start = time.time()

    def after_evaluation(self, evaluated_concept: Concept, *args, **kwargs):
        eval_time = time.time() - self._eval_start
        self._eval_time_total = self._eval_time_total + eval_time
        self._time_by_concept[evaluated_concept.name]["eval_time"] += eval_time

    def info(self) -> Dict[str, Any]:
        return {
            "time_evaluation_callback": {
                "time_by_concept": self._time_by_concept,
                "train_time_total": self._train_time_total,
                "eval_time_total": self._eval_time_total,
            }
        }
```

Callbacks: Concrete Implementation Example

```
class TimeEvaluationCallback(Callback, InfoProvider):
    def __init__(self):
        self._time_by_concept = defaultdict(lambda: dict({"train_time": 0, "eval_time": 0}))
        self._train_start = 0
        self._eval_start = 0
        self._train_time_total = 0
        self._eval_time_total = 0

    def before_training(self, *args, **kwargs):
        self._train_start = time.time()

    def after_training(self, learned_concept: Concept):
        train_time = time.time() - self._train_start
        self._time_by_concept[learned_concept.name]["train_time"] = train_time
        self._train_time_total = self._train_time_total + train_time

    def before_evaluation(self, *args, **kwargs):
        self._eval_start = time.time()

    def after_evaluation(self, evaluated_concept: Concept, *args, **kwargs):
        eval_time = time.time() - self._eval_start
        self._eval_time_total = self._eval_time_total + eval_time
        self._time_by_concept[evaluated_concept.name]["eval_time"] += eval_time

    def info(self) -> Dict[str, Any]:
        return {
            "time_evaluation_callback": {
                "time_by_concept": self._time_by_concept,
                "train_time_total": self._train_time_total,
                "eval_time_total": self._eval_time_total,
            }
        }
```

Strategy: Abstract Implementation Example

```
class Strategy(InfoProvider):
    """Base class for all continual learning strategies."""

    @abc.abstractmethod
    def name(self) -> str: ...

    def additional_info(self) -> Dict:
        return {}

    def info(self) -> Dict[str, Any]:
        return {"strategy": {"name": self.name(), **self.additional_info()}}
```

Strategy: Concrete Implementation Example

```
class ReplayOnlyStrategy(ConceptIncrementalStrategy, ConceptAwareStrategy):
    def __init__(self, model: Model, buffer: ReplayBuffer):
        self._model = model
        self._buffer = buffer

    def learn(self, data: np.ndarray, **kwargs) -> None:
        self._buffer.update(data)
        self._model.fit(self._buffer.data())

    def predict(self, data: np.ndarray, **kwargs) -> (np.ndarray, np.ndarray):
        return self._model.predict(data)

    def name(self) -> str:
        return "ReplayOnly"

    def additional_info(self) -> Dict:
        return {"replay_buffer": self._buffer.info()}
```

Strategy: Concrete Implementation Example

```
class ReplayOnlyStrategy(ConceptIncrementalStrategy, ConceptAwareStrategy):
    def __init__(self, model: Model, buffer: ReplayBuffer):
        self._model = model
        self._buffer = buffer

    def learn(self, data: np.ndarray, **kwargs) -> None:
        self._buffer.update(data)
        self._model.fit(self._buffer.data())

    def predict(self, data: np.ndarray, **kwargs) -> (np.ndarray, np.ndarray):
        return self._model.predict(data)

    def name(self) -> str:
        return "ReplayOnly"

    def additional_info(self) -> Dict:
        return {"replay_buffer": self._buffer.info()}
```

Practical Example

Let's repeat what we just learned and run our first experiment leveraging pyCLAD.

In this notebook, you will:

- Run your first experiment.
- Run the experiment involving real dataset.
- Have a chance to compare two different continual learning strategies.



https://github.com/lifelonglab/pyCLAD/blob/main/examples/getting_started.ipynb

Datasets and loaders

<https://huggingface.co/datasets/lifelonglab/>

Available datasets:

- UNSW
- NSL-KDD
- Wind Energy
- PV Energy

Single-line data loader:

```
dataset = UnswDataset(dataset_type="random_anomalies")
```

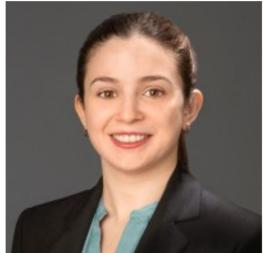


Call for Papers

Open World Anomaly Detection Workshop



Invited Speakers



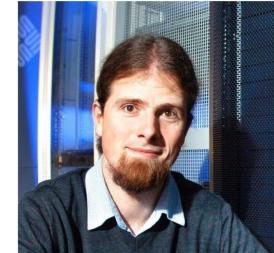
Camila Gonzalez

Stanford



Christopher Kanan

University of Rochester



Bartosz Krawczyk

Rochester Institute of Technology

Organizers



Roberto Corizzo

American University



Kamil Faber

AGH University of Krakow



Tyler L. Hayes

Georgia Institute of Technology

<https://sites.google.com/view/icdm2025-open-world-workshop>

Summarized takeaways

- **Continual anomaly detection** is an exciting avenue for research
 - New challenges, scenarios, metrics, etc.
- **Scenarios** can be created from any anomaly detection dataset of choice
- **Novel strategies are required** to fill a gap:
 - Current CL/LL strategies and real-world complexities
- **Task Agnostic CL/LL** is a more challenging/realistic learning setting.
 - Change detection can be adopted to trigger decision making in learning strategies

Thank you for your attention!

Questions?



Contacts

rcorizzo@american.edu



kfaber@agh.edu.pl



Linktree

<https://linktr.ee/lifelonglab>

