Applied Intelligence (2023) 53:29758–29780
https://doi.org/10.1007/s10489-023-05092-4

Exploring the potentials of online machine learning for predictive maintenance: a case study in the railway industry

Minh-Huong Le-Nguyen[1,2] · **Fabien Turgis**[2] · **Pierre-Emmanuel Fayemi**[2] · **Albert Bifet**[1]

Accepted: 6 October 2023 / Published online: 3 November 2023

Abstract

This study addresses data-driven predictive maintenance, an area in which machine learning has received considerable attention. Traditionally, a machine learning model is trained on static data before being put into production to predict failures on incoming data. However, new data typically present novelties that were not included in the training data, such as unexpected anomalies or faults. Such novelties reduce the model accuracy and require model retraining, which we consider to be a suboptimal practice.Therefore, we propose to leverage online machine learning as an adaptive and continuous alternative to implement efficient predictive maintenance on systems that produce data continuously. The literature on predictive maintenance concentrates primarily on failure prediction, whereas there are multiple stages in a standard predictive maintenance framework, such as data preprocessing and diagnostics, that require attention. In this study, we propose a modular pipeline consisting of three modules to execute many stages inside a predictive maintenance solution. Each module represents one of our original contributions. Firstly, because a system generates repeating patterns in the form of *cycles* when performing its functions, we construct an online active learning-based framework to extract these cycles from a stream of sensor data (cycle extraction with InterCE). Secondly, we implement an autoencoder for encoding the extracted cycles into *feature vectors* (feature learning with LSTM-AE). Thirdly, we develop an adaptive scoring function to compute the *health* of any system at any time using online clustering on the stream of feature vectors (health detection with CheMoc). These three contributions establish our framework for processing raw sensor data for predictive maintenance. We evaluate our methods using a real-world data set provided by SNCF, the French national railway company. For each experiment, we simulate a data stream consisting of sequentially arriving data from the provided data set to test our online algorithms. The experimental results demonstrate that (i) InterCE is able to extract cycles from a high-speed stream with greater accuracy than a hand-crafted expert system, (ii) LSTM-AE can identify meaningful features from the extracted cycles, and (iii) CheMoc can discover clusters that represent physical anomalies of the systems and capture the health evolution of the monitored systems. Due to a lack of ground-truth data at the time of writing, we have not implemented the prognostics method and will reserve this for future works. This study confirms the potential of online machine learning as an adaptive and lifelong learning solution for predictive maintenance.

Keywords Predictive maintenance · Online machine learning · Railway, data stream

1 Introduction

The railway facilitates long-distance mass transit and alleviates the burden of rush-hour traffic. As the most carbon-neutral mode of terrestrial mass transportation, the railway will continue to grow in the coming decades, thus demanding efficient maintenance.

In the railway, corrective maintenance and preventive maintenance are prevalent (Fig. 1). The former fixes a system after it fails, which incurs expensive costs and disrupts service; the latter conducts regular inspections to reduce the frequency of failures at the expense of a higher inspection cost.

Recently, a novel maintenance strategy known as predictive maintenance (PdM) has emerged [38]. PdM monitors the systems to predict critical failures, enabling near-corrective maintenance prior to the occurrence of a failure. PdM also identifies functional degradation and equipment maladjustment in order to optimize preventive maintenance.

✉ Minh-Huong Le-Nguyen
huong.le14895@gmail.com

Extended author information available on the last page of the article

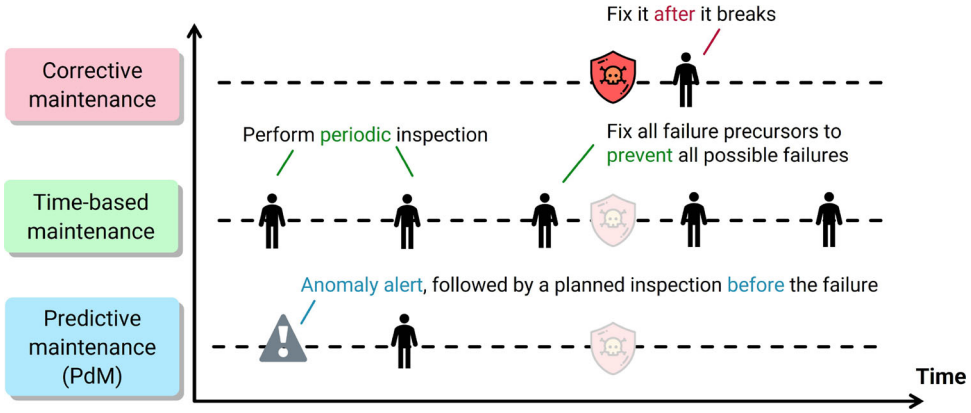


Fig. 1 Unlike corrective and time-based maintenance, predictive maintenance raises an alert only when an anomaly is detected, allowing for optimal maintenance planning

We focus on data-driven PdM, for which machine learning (ML) has become a major player [10]. ML comprises statistical learning algorithms that learn to perform a specific task on a set of observational data $\mathcal{D} = \{x_i, y_i\}_{i=1...N}$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$. An ML model is trained by learning a function $f(\mathcal{X}; \theta)$ such that the parameters θ are adjusted on $\mathcal{D}$ over multiple iterations until $f(\mathcal{X}; \theta)$ provides an accurate mapping from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. This is the principle of *supervised* learning, for which we always have one label y_i for each observation x_i. In the context of PdM, x_i can be a vector of sensor measurements recorded at a given time i, and y_i the state of the system at i (healthy, degraded, faulty, etc.). In some cases, there may not be any labels y_i, i.e., $\mathcal{D} = \{x_i\}_{i=1...N}$, possibly because a system is newly commissioned and its data are not yet labeled. This fits into the scenario of *unsupervised* learning, for which the main goal is to detect data structure via clustering or anomaly detection.

To use ML for PdM, it is standard practice to train an ML model on existing data, such as sensor signals, maintenance records, or past failure alerts. The trained model is then deployed in production to predict failures on incoming data. The model parameters θ do not change unless the model is retrained to learn θ from scratch. This is the *offline* approach to machine learning, as the model is trained on a fixed quantity of data and only begins making predictions after training is complete.

Although the offline approach has proven effective for many PdM applications [8], we deem it suboptimal, for the following reasons.

First of all, new generations of complex railway systems are typically outfitted with sensors that continuously record their physical measurements, thus producing an infinite data stream over time. Using the offline approach, we must sample data from the stream, but we do not know if this sample is representative of the stream's actual data distribution. If the majority of the data in the sample is produced by systems operating under normal conditions, a model trained on these ideal data will be inaccurate when used on data generated by degraded systems. As a result, the model must be retrained once the incoming data deviate largely from the data used for training. This is known as *data drifting* that commonly occurs in streaming data [14] and may manifest in different forms:

- concept drift (the relation between $\mathcal{X}$ and $\mathcal{Y}$ changes),
- concept evolution ($\mathcal{Y}$ has more possible values),
- feature drift ($\mathcal{X}$ changes, by changing its value range),
- feature evolution ($\mathcal{X}$ receives more or fewer measurement variables).

When data drifting occurs, the current model becomes obsolete and must be retrained on a new data set, which consists of either only new data collected since the occurrence of the drift or a combination of both new and old data. A model trained solely on new data will discard previous knowledge, but continuously adding new data to the training set is impractical due to the unboundedness of the data stream. In addition, training a complicated ML model can be computationally expensive.

Secondly, a data stream impedes the creation of *labels*. A label is the intended output value for an ML task and is individually assigned to every example. In a PdM application, a label may represent the estimated time until failure or the current condition of the system. To recognize the relationship between the target class and input variables, machine learning is notoriously data-hungry and requires a large number of training examples. In practice, labels are scarce for a variety of reasons: rare failure data (critical systems are frequently maintained and rarely fail), inaccurate labels, lack of human labelers, etc. This difficulty is exacerbated by the fact that manual labeling cannot keep up with the pace and volume of a data stream.

Also, in the railway, each occurrence of a failure is followed by the FRACAS[1] procedure to prevent it from re-occurring, and this failure label will not be given[2] again.

[1] Failure reporting, analysis, and corrective action system.

[2] It does not necessarily mean the failure will never occur again, but if it does, the domain experts will likely not label it again.

Fig. 2 Various stages in a condition-based/PdM solution [20]

Consequently, these issues impede the application of supervised learning techniques that require voluminous labeled data. We must therefore turn to unsupervised learning approaches, the goal of which is to identify insights or patterns in the incoming data.

Given that connected systems producing infinite amounts of data are already a reality on new generations of connected trains, the offline approach cannot keep up with this new horizon. Instead, *online machine learning* (OML) is an alternative learning paradigm that employs continuous learning to incrementally adjust a model based on new data. This allows for data drift adaptation. Its inherent incremental nature also enables OML to integrate a feedback loop to collect new feedback from humans, which serves as a label source.

The body of PdM literature strongly favors offline machine learning, and the few works that use online machine learning only concentrate on prognostics (failure prediction). Meanwhile, a PdM solution requires the implementation of multiple stages (Fig. 2). To the best of our knowledge, there is no prior work that employs online machine learning to implement all core PdM functionality from data preprocessing to failure prediction.

Therefore, our research question is **to investigate if OML could be used to implement the core functionality of a PdM solution for complex railway systems**. Our solution is a modular pipeline (Fig. 3) that accepts as input a data stream containing raw sensor signals from a fleet of M systems and outputs maintenance alerts to the technicians. Each module is implemented using online machine learning techniques and represents one of our contributions.

Our **first contribution** is the Interactive Cycle Extraction (InterCE) framework, which extracts operational *cycles* generated by any system as it executes its functions. Because a cycle reflects the condition of a system, it is of the utmost importance to accurately extract all cycles from the input data stream in order to prepare for the subsequent analysis. This module outputs a stream of cycles.

Our **second contribution** is a Long Short-term Memory Autoencoder (LSTM-AE) that learns to identify discriminative features from the extracted cycles. Although it is possible to analyze the cycles directly to determine the condition of a system, a cycle is inherently a multivariate time series, and time series analysis is typically time-consuming and resource-intensive, which conflicts with the requirement for efficient computation on a rapid data stream. Additionally, analyzing a sequence of time series (a stream of cycles) is even more challenging. Therefore, we simplify the problem by transforming each cycle into a *feature vector* that retains only the most relevant attributes of the original cycle. The output of this module is a stream of feature vectors.

Our **third contribution** is the Continuous Health Monitoring using Online Clustering (CheMoc) method, which uses online clustering to capture a set of collective health profiles of the systems as evolving clusters. A system's health profile may indicate either a normal or an abnormal condition. On the basis of these profiles, we then propose an adaptive scoring function for computing the *health score* of any system at any time. The output of this module is a stream of health scores.

We expect each method (InterCE, LSTM-AE, and CheMoc) to be more accurate than an expert system, which is a program created by a domain expert using fixed rules that perform the same tasks. Thus, an expert system represents human-level accuracy. Our objective is to demonstrate that online adaptive learning algorithms perform better than this baseline performance. The primary obstacles are the absence of labeled data

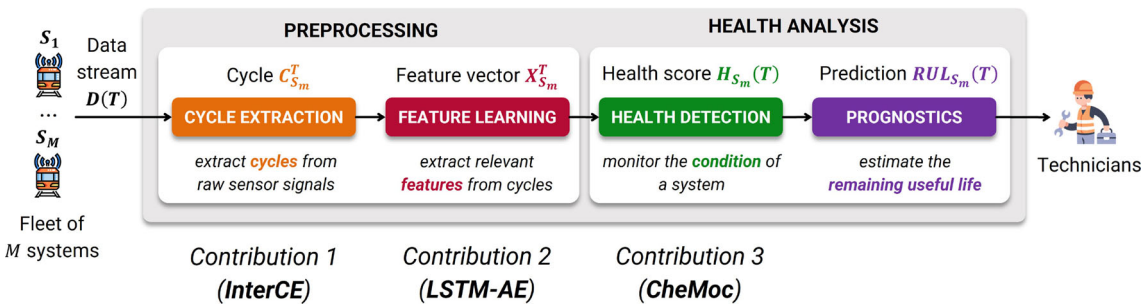


Fig. 3 A modular pipeline for PdM, where each module is implemented with OML

and, potentially, data drifting. Hence, our methods deal with dynamic and unlabeled streaming data in an unsupervised and incremental manner.

This article is organized as follows. Section 2 reviews the existing literature on the use of OML for PdM. Section 3 presents the case study we used to evaluate our methods on a representative class of complex railway systems. Section 4, 5, and 6 respectively describe InterCE, LSTM-AE, and CheMoc along with the experimental results of our case study. Section 7 concludes our work and outlines key research directions for future improvements.

2 Summary of existing literature

In the data-driven PdM literature, there is a strong preference for offline ML across many industries [16, 24, 32, 37]. Nonetheless, OML has begun to attract the interest of practitioners in the field.

The state of a system at a given time can be classified using supervised or semi-supervised techniques. Feng et al. [12] estimated the state of Li-ion batteries online by comparing new charging data to the support vectors learned offline on clean data from fresh battery cells, but their solution must be trained offline on a clean set of data before it can be deployed online to predict on incoming data. Ben Ali et al. [4] monitored the high-speed bearings of wind turbines using Adaptive Resonance Theory 2 (ART2) to categorize the turbines as healthy, degraded, or faulty. ART2 is online-compatible, but it does not offer a more granular view of system degradation. In addition, it assumes that a defined set of possible machine states is known in advance, whereas in reality, a machine may be affected by a number of unforeseen anomalies.

Anomaly detection is the most commonly used technique for dealing with unlabeled data, but it only detects the presence of an anomaly without providing additional information such as the nature or severity of an anomaly or monitoring how one or multiple anomalies impact a system. Aydemir and Acar [3] detected anomalies in machinery using a simple control chart monitoring called CUSUM and issuing an alert when signal values exceeded the predefined threshold for normal operations. However, CUSUM is too simplistic for complex systems because it only monitors a single performance parameter at a time. Tian et al. [34] performed online damage detection using one-class support vector machines to classify new data as normal or anomalous. Ribeiro et al. [28] predicted failures of automatic train doors using sequential anomaly detection, with an emphasis on the temporality of anomalies. Putina and Rossi [26] leveraged online clustering to detect anomalies in network telemetry data streams. Although not explicitly associated with PdM, their work entails anomaly detection, which is a component of machinery monitoring.

There are also works that emphasize the development of an online-compatible processing pipeline using offline ML algorithms. Canizo et al. [5] implemented a PdM application for wind turbines using various big data technologies, such as Apache Spark, HDFS, Apache Mesos, and Apache ZooKeeper, but the models were trained offline and did not adapt to new data. Su and Huang [33] predicted hard-disk drive failures in data centers using Apache Spark for real-time data analytics and Hadoop for batch processing. The models were trained offline using random forest and remained constant despite the arrival of novelties in the data stream. Sahal et al. [30] reviewed open-source big data solutions and their applicability to PdM, exemplified by two case studies on railway and wind turbine maintenance.

In general, works in this vein either propose novel OML algorithms to address a single functionality of a PdM framework (primarily, failure prediction) or adapt existing methods to implement a streaming-based solution for PdM. Our research tends towards the latter. Rather than proposing entirely novel algorithms, we combine existing techniques and algorithms to address the core functionality of a PdM application: data preprocessing with InterCE and LSTM-AE, and diagnostics with CheMoc. When necessary, we modify existing OML algorithms to align them with railway operational constraints, while maintaining their accuracy and effectiveness on data streams.

3 Case study: the passenger access systems

We design and evaluate our methods using sensor data from the passenger access system (PAS), which is the automatic train door system on a fleet of passenger trains. The data set is supplied by a French national railway company. We chose the PAS because its complexity is reflective of many other systems in the railway. If we are able to develop a viable PdM solution for the PAS using OML, we will be able to apply OML to other types of complex railway systems.

A PAS enables passengers to enter and exit the train by opening and closing at each station stop. Each PAS is equipped with a control unit that processes the train's central commands and records sensor signals. When a train enters a station and all PASs receive the opening authorization command, the data acquisition begins. All PAS signals are recorded until the PASs are completely closed and locked, and the train is ready to depart. The signals are encoded and written into a binary file. Each PAS produces a single file (Fig. 4). This fleet of passenger trains contains 7000 PASs, each of which opens and closes every three minutes.

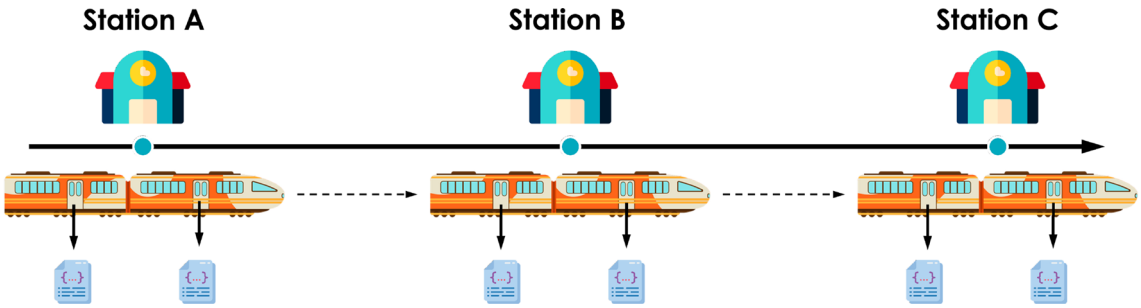


Fig. 4 Every time a train enters and leaves a station, each PAS produces one file

Each file contains a multivariate time series with both analog and boolean variables. There are over 60 boolean variables and three analog variables associated with the electric motor's position, voltage, and current intensity in a PAS. The PASs on a train in operation continuously produce data. Over time, a fleet of multiple trains generates a fast and infinite data stream comprised of sensor signals from thousands of PASs. This scenario makes OML a suitable approach to enable continuous learning.

Our methods are designed to work with the data originating from any systems in the same fleet. Please note that we only analyze the data from systems of the same type (e.g., all PASs) and do not combine data from various types of systems (e.g., PASs and batteries). We denote $\mathcal{S} = \{S_1, \ldots, S_M\}$ the fleet of M systems being monitored, where $S_m \in \mathcal{S}$ is the ID of a system (for example, `train 1/car 2/left door`).

4 Cycle extraction

A PAS exhibits cyclic behavior because it executes its functions repeatedly. When a train enters a station, a PAS opens and closes to allow passengers to embark and exit. Each opening and closing of the PAS generates a *cycle* that records the behavior of a PAS when it performs a function. The underlying condition of the PAS can be revealed through these cycles. The first task is to extract all cycles from the sensor data stream.

A cycle $C^T_{S_m}$ created by a system $S_m \in \mathcal{S}$ at a time T is a sequence of N timesteps, such that each timestep $\mathbf{a}^{(i)}{}_{1 \leqslant i \leqslant N}$ is a vector of D variables. In other words, a cycle is a multivariate time series, in which each timestep is a record of multiple measurements.

$$C^T_{S_m} = [\,\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(N)}\,] \quad \text{where} \quad \mathbf{a}^{(i)} \in \mathbb{R}^D \tag{1}$$

Figure 5 illustrates an example where two cycles, an opening and a closing of a PAS, are extracted from an input sequence $\mathbf{s} = [\,\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(N')}\,]$ $(N' \leqslant N)$ containing raw sensor signals from a system S_m. Each input sequence comes from one data file produced by one PAS when the train enters a station (Fig. 4).

Cycle extraction involves two steps: cycle detection and identification. The former detects the presence of a cycle in raw signals, while the latter assigns the detected cycle to a system function, such as "door opening" or "door closing".

4.1 InterCE

Several techniques exist for online cycle extraction, including change point detection [2], motif discovery [35], and discretization [22]. All of these techniques, however, have the same flaw: they detect cycles based on statistical changes or detectable patterns in the data, but they disregard the semantics of the cycles. A cycle is not merely a recurring motif; rather, it must represent a known function of the system, which cannot be learned from sensor data alone. Therefore,

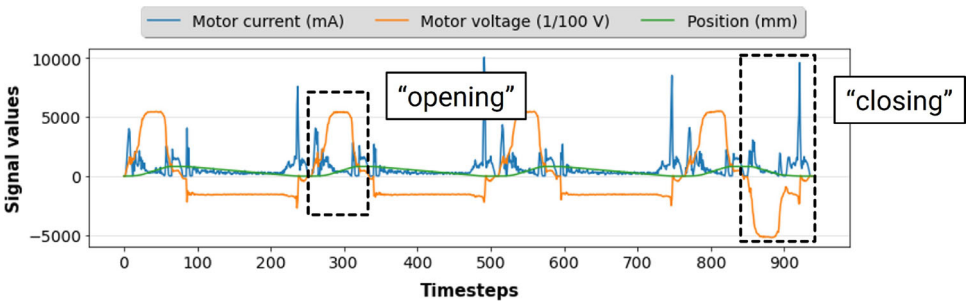


Fig. 5 Example of two cycles extracted from an input file (not exhaustive)

human understanding of the anticipated cycles must be incorporated into the learning process.

From this observation, we develop the framework InterCE (Interactive Cycle Extraction) based on the active learning paradigm [31] to query for human feedback on inputs that InterCE does not know how to process, and extracts cycles automatically otherwise (Algorithm 1). Given a new input sequence $\mathbf{s}$, InterCE first feeds it to an ensemble $\mathcal{E}$ containing various cycle detection algorithms to obtain the candidate cycles $\mathbb{C}(\mathbf{s})$ (line 1). Then, InterCE's memory $\mathcal{M}$ checks if $\mathbf{s}$ is similar to an input sequence that InterCE has processed in the past (and thus already has the knowledge to extract cycles from similar inputs). If such is the case, InterCE uses its knowledge $\mathcal{K}$ to automatically detect and label cycles (line 2-4). Otherwise, if $\mathbf{s}$ is a sequence with novel motifs, InterCE sends a new query $Qr(\mathbf{s})$ to a domain expert (line 7-9).

Algorithm 1 InterCE (managing queries).

Input: A sequence $\mathbf{s} = [\,\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(N')}\,]$
Output: The extracted cycles $\hat{\mathcal{C}}(\mathbf{s})$ or a query $Qr(\mathbf{s})$

1 $\mathbb{C}(\mathbf{s}) \leftarrow \mathcal{E}$.extract($\mathbf{s}$) /* get the candidates from $\mathbf{s}$ */
2 **if** $\mathcal{M}$.hasSeen($\mathbf{s}$) *is True* **then** /* whether $\mathbf{s}$ is a novel motif */
3 **if** $\mathcal{M}$.hasProcessedQueryOf($\mathbf{s}$) *is True* **then**
4 $\hat{\mathcal{C}}(\mathbf{s}) \leftarrow \mathcal{K}$.selectCycles($\mathbb{C}(\mathbf{s})$)
5 **else** /* redundant query, buffering instead of sending */
6 $Qr(\mathbf{s}) \leftarrow \mathcal{M}$.createBufferedQuery($\mathbf{s}$, $\mathbb{C}(\mathbf{s})$)
7 **else** /* create a new feedback to the human */
8 $Qr(\mathbf{s}) \leftarrow \mathcal{M}$.createOfficialQuery($\mathbf{s}$, $\mathbb{C}(\mathbf{s})$)
9 sendToHuman($Qr(\mathbf{s})$)

However, it may happen that, while waiting for the feedback of a query $Qr(\mathbf{s})$, another sequence $\mathbf{s}'$ that is very similar to $\mathbf{s}$ arrives from the data stream. At this moment, InterCE has not received the feedback to $Qr(\mathbf{s})$, therefore it does not have the knowledge to process $\mathbf{s}'$ and will create a query $Qr(\mathbf{s}')$ that is redundant to $Qr(\mathbf{s})$. To avoid redundant queries, we distinguish between official queries (line 8) that are sent to the humans, and buffered queries that are very similar to an official query but are queued in line rather than sent to the humans (line 5-6).

In parallel, InterCE continuously listens for new feedback (Algorithm 2). A feedback contains the cycles that the human wants to keep from the candidates $\mathcal{C}(\mathbf{s})$. When a feedback for a query $Qr(\mathbf{s})$ is received from the domain expert, first, InterCE uses it to solve the official query $Qr(\mathbf{s})$ (line 2). Then, InterCE retrieves all the buffered queries $\mathcal{Q}$ that are linked to the official query $Qr(\mathbf{s})$ and solves them in bulk using the same feedback (line 3-5).

Algorithm 2 InterCE (processing feedback).

Output : The cycles to an official query $Qr(\mathbf{s})$ and to the linked buffered queries
// infinitely listening for new human feedback
1 **while** *new feedback* $Fb(Qr(\mathbf{s}))$ *becomes available* **do**
  // select the cycles from $\mathbb{C}(\mathbf{s})$ as instructed by $Fb(Qr(\mathbf{s}))$
2 $\hat{\mathcal{C}}(\mathbf{s}) \leftarrow \mathcal{K}$.solveOfficialQuery($Qr(\mathbf{s})$);
  // use $Fb(Qr(\mathbf{s}))$ to process the buffered queries of $Qr(\mathbf{s})$
3 $\mathcal{Q} \leftarrow \mathcal{K}$.getBufferedQueriesOf($Qr(\mathbf{s})$);
4 **foreach** $Qr(\mathbf{s}') \in \mathcal{Q}$ **do**
5 $\hat{\mathcal{C}}(\mathbf{s}') \leftarrow \mathcal{K}$.selectCycles($Qr(\mathbf{s}')$)

Ensemble of extractors

Rather than using a single extractor, we employ an ensemble of J extractors $\mathcal{E} = [\,E_1, \ldots, E_J\,]$ to make InterCE more robust against noises through ensemble learning [25], because preliminary findings suggest that a single extractor cannot always capture all intended cycles (Fig. 6). We develop three extractors for the ensemble $\mathcal{E}$. This supplementary material contains the complete description of the extractors [19].

Let $\mathcal{C}_{E_j}(\mathbf{s}) = \{\, C_{S_m}^{(j,1)}(\mathbf{s}), \ldots, C_{S_m}^{(j,n_j)}(\mathbf{s}) \,\}$ be the set of cycles extracted from $\mathbf{s}$ by an extractor $E_j \in \mathcal{E}$, for instance, $C_{S_m}^{(j,1)}(\mathbf{s})$ denotes the first cycle detected by the extractor E_j from the input sequence $\mathbf{s}$. The candidates $\mathbb{C}(\mathbf{s})$ extracted by all the extractors in $\mathcal{E}$ from $\mathbf{s}$ is thus $\mathbb{C}(\mathbf{s}) = \left\{\, \mathcal{C}_{E_1}(\mathbf{s}), \ldots, \mathcal{C}_{E_J}(\mathbf{s}) \,\right\}$. The next step is to retain only the most correct cycles from the candidates $\mathcal{C}(\mathbf{s})$

Memory

The memory $\mathcal{M}$ stores a set of unique data motifs, denoted $\mathcal{P}$, to determine whether an input $\mathbf{s}$ can be automatically processed or if a query must be issued. Because a system performs a fixed number of functions, inputs containing a repeating pattern are frequently present in the data. Commonly, a PAS opens and closes when a train stops at a station; thus, there are multiple input sequences with only an opening cycle followed by a closing cycle.

In light of this observation, and in an effort to reduce the number of queries, InterCE issues queries only for novel data motifs. A data motif is the entirety of an input sequence's series. Additionally, InterCE stores each novel data motif in

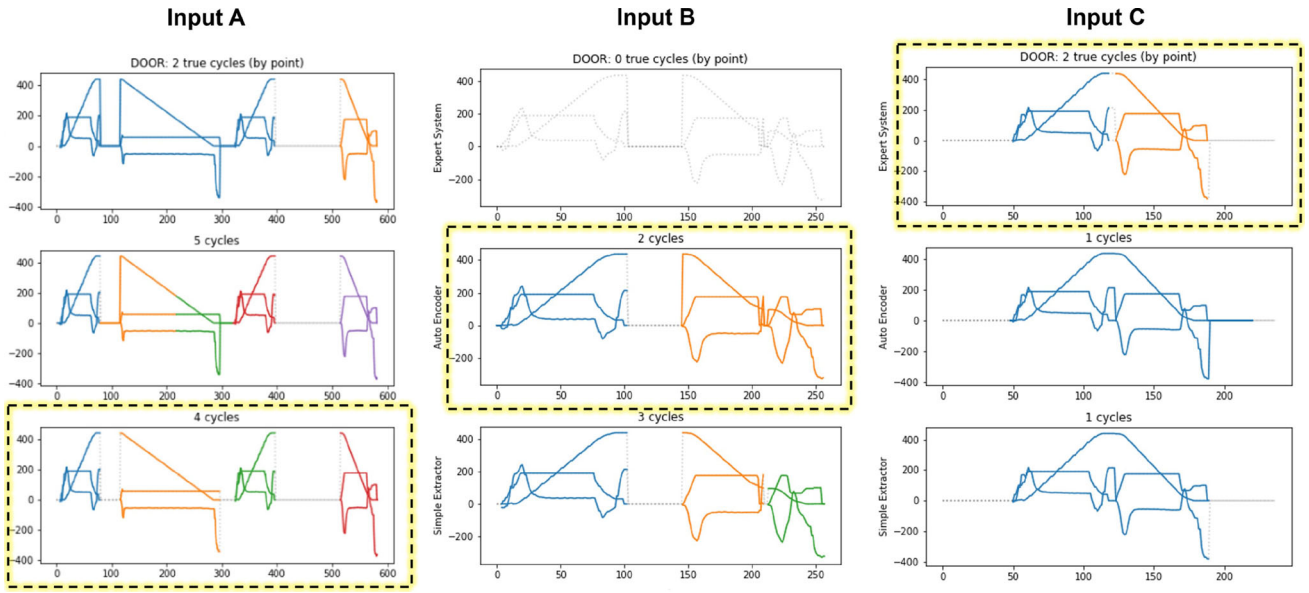


Fig. 6 On various inputs, the same extractor does not always produce accurate results. Each column represents an input sequence, and each row represents the extraction results of an extractor applied to each sequence. The highlighted boxes indicate the optimal extraction for each input

$\mathcal{P}$. For instance, in Fig. 6, the three input sequences are considered three distinct motifs because they differ substantially.

Knowledge

The knowledge $\mathcal{K}$ maintains the feedback as clusters and automatically selects cycles from $\mathbb{C}(X)$ when the memory $\mathcal{M}$ determines that a new query is not necessary (i.e., InterCE has already learned how to process an input $\mathbf{s}$). To learn the typical shape of a function, we cluster the cycles based on their label. Only the cycles from human feedback are used to build the clusters. To avoid undesirable clustering errors, we refrain from using an algorithm to automatically group the feedback cycles. Instead, we assume that human experts always provide accurate feedback and we simply categorize the cycles in the feedback $Fb(Qr(\mathbf{s}))$ by their label. Consequently, all cycles with the same label are put in the same cluster. Ultimately, the number of clusters equals the number of functions a system is capable of performing. For the PASs, there are two clusters: one for door opening and one for door closing.

Upon receiving new cycles, the centroid of a cluster is recalculated by averaging across the timesteps of all cycles in this cluster. We store the clusters directly in memory to enable fast access when $\mathcal{K}$ selects the cycles. This could pose a storage problem as InterCE works on an infinite stream. Nevertheless, because the systems perform a limited number of functions and generate a limited number of unique cycle types, the clusters should be kept at a reasonable size.

Algorithm 14 describes how $\mathcal{K}$ selects cycles from the candidates $\mathbb{C}(\mathbf{s})$. First, $\mathcal{K}$ computes the average distance of all the cycles by each extractor $E_j \in \mathcal{E}$ to the feedback clusters (line 1-7). The extractor with the lowest average distance is considered the best one, and all the cycles it proposes are selected as the most correct cycles $\hat{\mathcal{C}}(\mathbf{s})$ (line 8). The cycles are labeled with the label of their closest cluster (line 10-13). For now, $\mathcal{K}$ selects the cycles proposed by a single extractor. In the future, we will allow $\mathcal{K}$ to select cycles from different extractors.

Algorithm 3 Selecting cycles from $\mathbb{C}(\mathbf{s})$ automatically.

Data: The candidates $\mathbb{C}(\mathbf{s})$
Output : The final cycles $\hat{\mathcal{C}}(\mathbf{s})$

1 $dist \leftarrow \emptyset$;
2 **foreach** *candidate* $\mathcal{C}_{E_j}(\mathbf{s})$ **do**
3 $\quad dist[E_j] \leftarrow 0$;
4 $\quad$ **foreach** *cycle* $C^{(j,i)}_{S_m}(\mathbf{s})$ *in* $\mathcal{C}_{E_j}(\mathbf{s})$ **do**
5 $\quad\quad c \leftarrow K.\texttt{findClosestCluster}(C^{(j,i)}_{S_m}(\mathbf{s}))$;
6 $\quad\quad dist[E_j] \leftarrow dist[E_j] + \texttt{distance}(C^{(j,i)}_{S_m}, c)$;
7 $\quad dist[E_j] \leftarrow \frac{dist[E_j]}{J}$;
8 $\hat{\mathcal{C}}(\mathbf{s}) \leftarrow \text{argmin } dist, \quad \forall \mathcal{C}_{E_j} \in \mathbb{C}(\mathbf{s})$;
9 $R \leftarrow \emptyset$;
10 **foreach** *cycle* $C^T_{S_m} \in \hat{\mathcal{C}}(\mathbf{s})$ **do**
11 $\quad c \leftarrow K.\texttt{findClosestCluster}(C^T_{S_m}(\mathbf{s}))$;
12 $\quad y^T_{S_m} \leftarrow c.\text{getLabel}()$;
13 $\quad R \leftarrow R \cup \{(C^T_{S_m}, y^T_{S_m})\}$;
14 **return** R

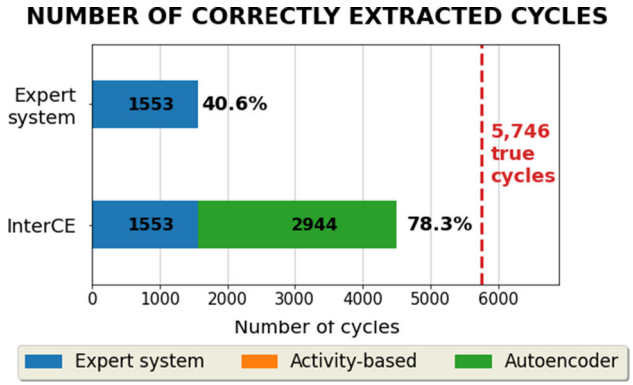


Fig. 7 Number of cycles correctly extracted by the expert system and by InterCE

4.2 Experimental results

To evaluate InterCE, we compare its extraction accuracy to that of a hand-crafted expert system for the PAS data set. We include the expert system as a baseline performance in the ensemble $\mathcal{E}$ of InterCE. We expect that InterCE will substantially outperform the expert system (Section 4.2.1). We also analyze InterCE's processing time per input and query ratio to determine its online compatibility (Section 4.2.2).

4.2.1 Extraction accuracy

We set up an experiment with 1000 data files and a human actively answering InterCE's queries. Then, we manually inspect the extraction results of these files and record the number of correct cycles by InterCE and by the expert system. The accuracy score of each method is calculated by dividing the number of cycles correctly extracted by the total number of cycles that must be extracted (ground-truth).

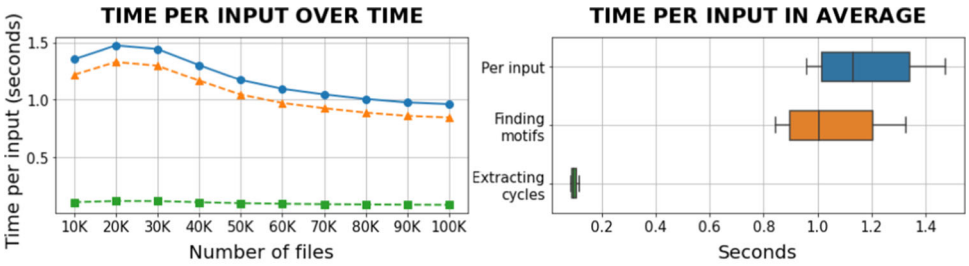
Figure 7 shows the accuracy of InterCE and of the expert system, along with the number of correct cycles by individual extractors in $\mathcal{E}$. Because InterCE integrates the expert system in $\mathcal{E}$, it naturally achieves the accuracy level of the expert system. The autoencoder-based extractor significantly improves the accuracy of InterCE. The activity-based extractor never selects the optimal cycles due to its inability to recognize a variety of input sequence patterns due to its simplicity.

4.2.2 Efficiency

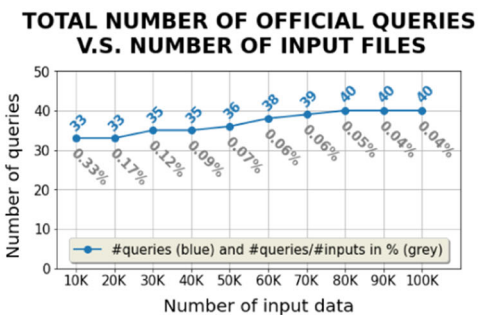
We evaluate the efficiency of InterCE based on the number of official queries and the processing time by executing InterCE on 100,000 input files without providing any feedback. Figure 8a illustrates the execution time per input file. The time required by $\mathcal{M}$ to identify similar motifs predominates over the time required by $\mathcal{E}$ to extract cycles, which is negligible. The time required to identify motifs decreases as the number of novel motifs stabilizes.

Figure 8b depicts the number of queries and the ratio of the number of queries to the number of input files. The number of queries is negligible relative to the number of input sequences, with only 40 queries issued over 100,000 input files (0.04% of total queries). As the growth rate of the volume of input sequences is significantly greater than that of the number of queries, the query ratio decreases over time and approaches zero as the number of inputs approaches infinity.

Fig. 8 Efficiency evaluation of InterCE



(a) Execution time per input file



(b) Number of official queries created

Discussion

The results demonstrate that InterCE substantially improves the expert system's accuracy and achieves a processing rate of one file per second with a decreasing query ratio. However, InterCE has two major drawbacks. First, although InterCE is updatable on human feedback, it only adapts the knowledge $\mathcal{K}$ and not the extractors in the ensemble $\mathcal{E}$, which would be advantageous for enhancing the overall detection accuracy. Second, as the number of motifs increases, the bottleneck of motif matching will deteriorate. The accuracy of motif matching is also important in order to avoid redundant queries.

These issues stem from the fact that InterCE *memorizes* the data rather than truly *learning* from them. Instead of memorizing the data motifs, the memory $\mathcal{M}$ could learn the features of these motifs to quickly determine if a motif has been encountered, thereby re-framing motif matching as a classification problem rather than a search problem. Then, instead of merely storing the cycle shapes and labels, InterCE could use a classifier and/or a clustering algorithm to perform a more accurate mapping in $\mathcal{K}$. In particular, the learners in $\mathcal{E}$ should be updatable. Either we enable InterCE to update the extractors' hyperparameters on new feedback, or the extractors themselves must adapt to changes in the stream.

5 Feature learning

After extracting the cycles, we can analyze them to determine the current condition of the systems. However, it is inefficient to analyze the cycles as time series. Element-wise comparison of millions of cycles is computationally expensive, whereas a cycle can be summarized by retaining only its most significant *features*, such as cycle length, average value, and skewness. Therefore, we transform each cycle $C^T_{S_m}$ to a *feature vector* $X^T_{S_m}$ of P real-valued numbers (2).

$$X^T_{S_m} = (x_1, ..., x_P) \in \mathbb{R}^P \tag{2}$$

We implement a long short-term memory autoencoder (LSTM-AE) to derive representative features from the cycles. An autoencoder is a neural network that is incrementally updatable with new data, making it appropriate for online learning. From each cycle $C^T_{S_m}$ of N timesteps and D variables, the LSTM-AE derives a feature vector $X^T_{S_m}$ of P variables, such that $P \ll N \times D$ to ensure compactness.

5.1 LSTM-AE

An autoencoder is a neural network that learns to reconstruct its own input. It is composed of an encoder f and a decoder g with weights θ_f and θ_g respectively (Fig. 9). We denote $\boldsymbol{\theta} = [\theta_f, \theta_g]$ the weights of the entire network.

The encoder receives an input x and feeds it to the next layers, compressing x through the layers via non-linear transformations. The last layer of the encoder produces the final features $h = f(x; \theta_f)$. The decoder passes h through its own layers to reconstruct x at the final layer, i.e., $\hat{x} = g(h; \theta_g) = g(f(x; \theta_f); \theta_g)$. The reconstruction error between x and $\hat{x}$ is propagated backward in the network to update $\boldsymbol{\theta}$ in order to reduce the reconstruction error in the next iteration. Training an autoencoder means adjusting $\boldsymbol{\theta}$ iteratively to optimize an objective function. Typically, the objective function is the mean of the reconstruction errors (mean squared errors) over all the n training examples.

$$\begin{aligned} J(\boldsymbol{\theta}; X) &= \sum_{i=1}^{n} \left(x^{(i)} - \hat{x}^{(i)}\right)^2 \\ &= \sum_{i=1}^{n} \left(x^{(i)} - g(f(x^{(i)}; \theta_f); \theta_g)\right)^2 \end{aligned} \tag{3}$$

The weights $\boldsymbol{\theta}$ are adjusted by the backpropagation algorithm. After each iteration, the weights take one gradient step toward the direction of the optimum, using the gradient calculated from the objective function. In (4), η is the learning rate, dictating how large a weight update step should be with respect to the gradient of the cost $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X)$. When the reconstruction errors cannot be further reduced, the autoencoder has found the optimal weights that allow it to identify representative features.

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X) \tag{4}$$

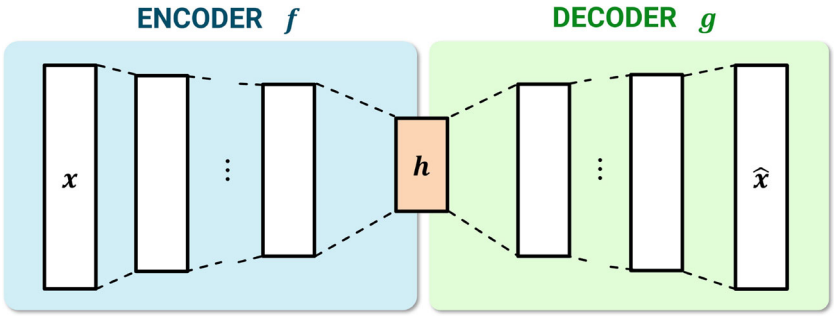


Fig. 9 A basic autoencoder

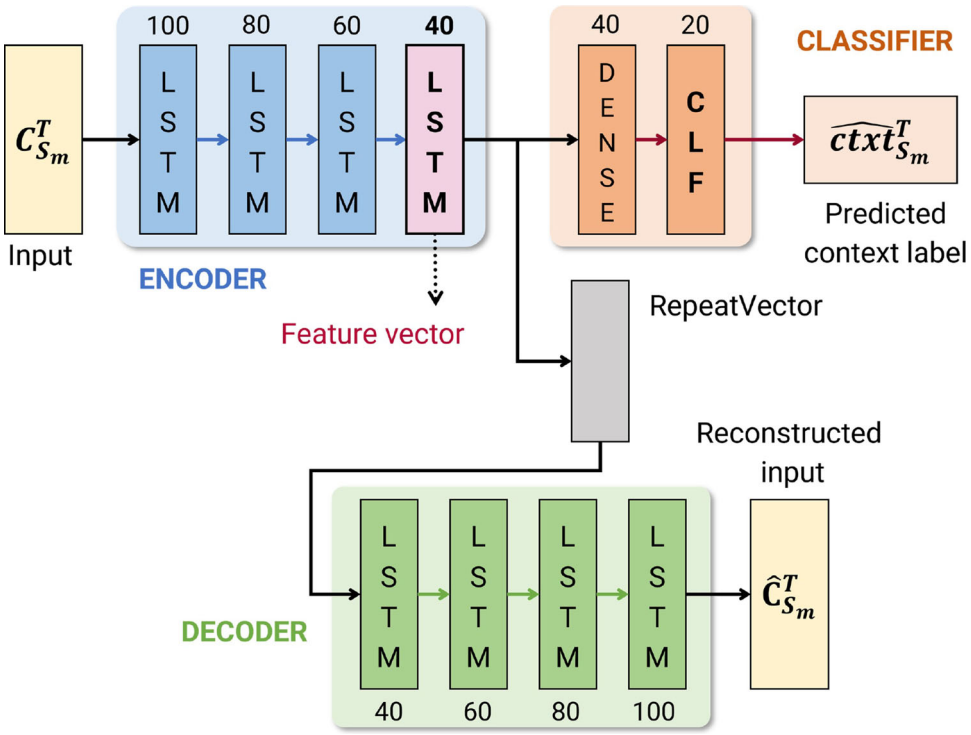


Fig. 10 The joint architecture of the LSTM-AE that connects the decoder and the context classifier to the encoded features

We choose a recurrent architecture for handling sequential data as we work with time series (cycles). Among existing recurrent architectures, we select the long short-term memory architecture (LSTM) [15] because our experiments show that the LSTM is capable of overcoming the exploding gradient phenomenon. We implement a symmetrical architecture with a final encoder layer size of 40 (Fig. 10). We opt for this architecture empirically by comparing the reconstruction capacity of various architectures with different numbers of layers and layer sizes, and then selecting the architecture with the best reconstructions.

Furthermore, the operational context, such as the outside temperature or the train station, can cause noises in the cycles, which make normal cycles appear abnormal. To make the LSTM-AE robust against contextual noises, we construct a joint architecture with a classifier g_c of weight θ_c that maps a cycle to its own context (Fig. 10). The intuition is that the autoencoder learns to derive features from a cycle that are representative (good reconstruction) and map well to the context in which this cycle was generated (good context recognition). The encoder's weight θ_f is adjusted by both the reconstruction accuracy and the context classification accuracy.

According to a railway expert, the train stations are a significant context factor that produces noises in the cycles. Thus, we choose the train stations as the target label for classification. Since the station information is always available for each cycle, the labels necessary to train the classifier are always available. Therefore, no annotation effort is needed.

Let $\mathcal{L}_g$ be the loss function of the decoder, $\mathcal{L}_c$ the loss function of the classifier, n the number of training examples, P the dimension of the feature vectors, and C the number of class labels (the number of train stations), we use the mean squared error (MSE) for $\mathcal{L}_g$ because the decoder outputs real-valued vectors, and categorical cross-entropy (CCE) for $\mathcal{L}_c$ to optimize θ_c for multiclass classification. Let $x^{(i)}$ be the i^{th} training cycle, $\hat{x}^{(i)}$ its reconstruction, $y^{(i)}$ the true station of the i^{th} training cycle, $\hat{y}^{(i)}$ the classification output.

The original formula of the MSE is:

$$MSE(x_i, \hat{x}_i) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

but because a cycle is a multivariate time series, the MSE of a cycle and its reconstruction is the average of the MSE of each univariate series in the cycle, and the MSE of each univariate series is the averaged MSE over N timesteps[3] in the cycle (5). The CCE loss is computed on the probability $p_c^{(i)}$ associated to each c^{th} train station of the i^{th} example $(1 \leqslant c \leqslant C)$ (6).

$$\mathcal{L}_g\left(x^{(i)}, \hat{x}^{(i)}\right) = \frac{1}{D}\sum_{j=1}^{D} MSE_j^{(i)}$$

[3] As cycles may differ slightly in length, we pad them with 0.0 to have equal-length cycles.

$$= \frac{1}{DN} \sum_{j=1}^{D} \sum_{k=1}^{N} \left(x_{j,k}^{(i)} - \hat{x}_{j,k}^{(i)} \right)^2 \quad (5)$$

$$\mathcal{L}_c \left(y^{(i)}, \hat{y}^{(i)} \right) = - \sum_{c=1}^{C} p_c^{(i)} \log \hat{p}_c^{(i)} \quad (6)$$

To find the optimal parameters $\boldsymbol{\theta} = [\theta_f, \theta_g, \theta_c]$ of the LSTM-AE, the objective function is the sum of the MSE and CCE losses over all the training examples (7). To update the weights $\boldsymbol{\theta}$ at each iteration, the gradient used for backpropagation is the gradient of the sum of the two losses, and by linearity, is the sum of the gradient of each loss. The decoder and classifier are used to optimize the weights $\boldsymbol{\theta}$, but we only use the encoder to extract features from cycles after training.

$$J(\boldsymbol{\theta}; X, y) = \frac{1}{n} \sum_{i=1}^{n} \left[\mathcal{L}_g(x^{(i)}, \hat{x}^{(i)}) + \mathcal{L}_c(y^{(i)}, \hat{y}^{(i)}) \right] \quad (7)$$

5.2 Experimental results

The LSTM-AE is implemented with `tensorflow` [1]. The training set contains clean cycles. The testing set has both noisy and clean cycles. We train one LSTM-AE for each cycle type and independently evaluate the features of each type. Table 1 shows the size of the training and testing sets of each cycle type.

To evaluate the features learned by the LSTM-AE, we compare them to the indicators identified by the expert system (Table 2). The expert indicators include statistics that can be extracted from a cycle, such as the mean, the standard deviation, and the minimum/maximum values. Because the cycles cannot be reconstructed from the expert indicators, we cannot directly measure the amount of information loss to evaluate the quality of the features. Instead, we conduct a *ranking* evaluation: if the features are learned accurately from the cycles, anomalous cycles will produce anomalous features, while normal cycles will produce normal features. As a result, the top-k most anomalous cycles must correspond to the top-k most anomalous feature vectors.

Let $\mathcal{C}_k = [\, C^{(1)}, \ldots, C^{(k)} \,]$ and $\mathcal{X}_k = [\, X^{(1)}, \ldots, X^{(k)} \,]$ be the ranking of the top k most anomalous cycles and top k most anomalous feature vectors respectively, such that for all $i < j$, $C^{(i)}$ is more anomalous than $C^{(j)}$, and similarly for $X^{(i)}$ and $X^{(j)}$. Our goal is to verify whether the ordering of $\mathcal{C}_k$ and of $\mathcal{X}_k$ match, i.e., $\forall i \in [1, k]$, $X^{(i)}$ is indeed the feature vector derived from $C^{(i)}$. We denote the ranking of the expert indicators[4] as $\mathcal{X}_k^E$ and that of the LSTM-AE features as $\mathcal{X}_k^L$.

Let $\overline{C}$ be the profile[5] of clean cycles used to train the LSTM-AE, $\overline{X}^E$ and $\overline{X}^L$ the profile of clean expert indicators and of clean LSTM-AE features extracted from the same set of clean cycles. The ranking of the cycles, of the expert indicators, and of the LSTM-AE features are obtained as follows.

- The abnormality of a cycle is based on the area under the curves[6] (AUC) of this cycle to $\overline{C}$ (greater area means higher abnormality).
- The abnormality of an expert indicator vector is the Euclidean distance between this vector to $\overline{X}^E$ (greater distance means higher abnormality).
- The abnormality of an LSTM-AE feature vector is the Euclidean distance between this vector to $\overline{X}^L$ (greater distance means higher abnormality).

Figure 11 explains the evaluation setting. The ground-truth ranking $\mathcal{C}$ is obtained by sorting the cycles in the test set C_{test} by their AUC to $\overline{C}$ in the descending order. The ranking of the LSTM-AE features $\mathcal{X}_k^L$ is obtained by the distance of the test feature vectors $\hat{X}^L$ to $\overline{X}^L$, and similarly for $\mathcal{X}_k^E$ to $\hat{X}^E$ and $\overline{X}^E$. The LSTM-AE features are better at information preservation than the expert indicators if $\mathcal{X}_k^L$ matches with $\mathcal{C}$ better than $\mathcal{X}_k^E$ does.

To compare two rankings, we use the normalized discounted cumulative gain (nDCG) [17] that evaluates an algorithm-based ranking with respect to a ground-truth ranking by a score in the interval [0, 1], with a score of 1 indicating the perfect match. The nDCG is frequently used in information retrieval: an item with a high relevance score (its *gain*)

Table 1 Number of training and testing cycles

Door opening	Training	13,456
	Testing	21,181
Door closing	Training	14,224
	Testing	11,851

Table 2 Size of the feature vectors and expert indicator vectors

Door opening	LSTM-AE	40
	Expert indicators	83
Door closing	LSTM-AE	40
	Expert indicators	70

[4] In this context, an expert indicator is short for a vector of expert indicators, and similarly for the LSTM-AE feature vectors and features.

[5] The profile of a set of cycles is the average computed at each timestep over all the cycles.

[6] A curve is a univariate time series in a cycle. The AUC of one cycle is the average of the areas of all its univariate series.

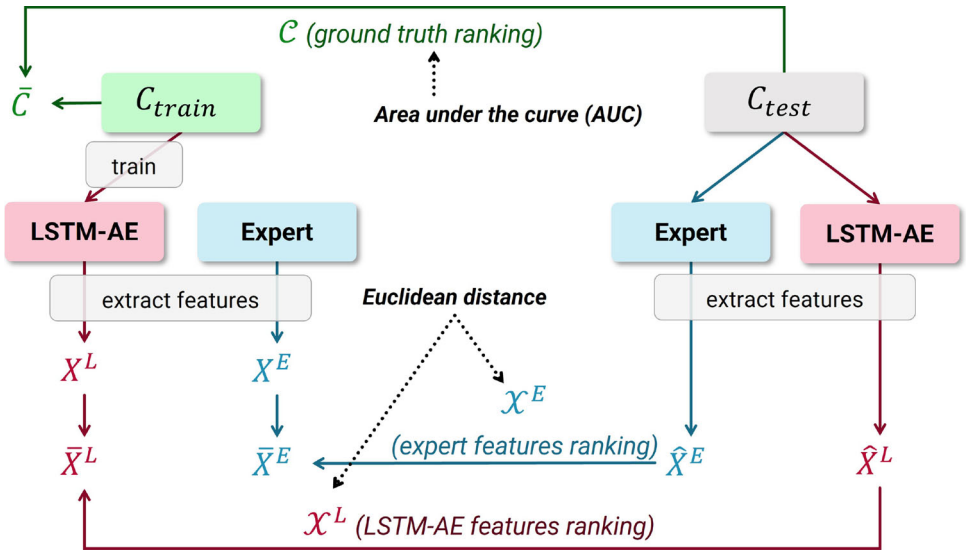


Fig. 11 Experiment setting of the cycle-feature ranking evaluation

is more relevant to the user's query and should be put near the top of the ranking.

The discounted cumulative gain (DCG) of the top k items is computed by summing, for each position from 1 to k, their relevance score divided by a logarithmic rank-based discount factor (8). The ideal discounted cumulative gain (iDCG) is calculated similarly, but the relevance score G_i at each position i is replaced by the relevance score of the ground-truth ranking. The iDCG returns the highest possible gain of the correct ranking. The nDCG is obtained by dividing DCG by iDCG.

$$DCG@k = \sum_{i=1}^{k} \frac{G_i}{\log_2(i+1)} \tag{8}$$

We rank the test cycles C_{test} by their AUC to $\overline{C}$, the test indicator vectors $\hat{X}^E$ and the test feature vectors $\hat{X}^L$ by their Euclidean distance to $\overline{X}^E$ and $\overline{X}^L$, respectively. Once we have the ranking $\mathcal{C}$, $\mathcal{X}_k^L$, and $\mathcal{X}_k^E$, we use the AUC of the cycles as the relevance scores for the nDCG. The ideal relevance scores are the ranked AUC of the cycles in $\overline{C}$. Then, we compare nDCG@k($\mathcal{X}^E, \mathcal{C}$) to nDCG@k($\mathcal{X}^L, \mathcal{C}$) on $k = [100, ..., 1000]$ (Fig. 12, top row) and on $k = [1000, ..., 10000]$ (Fig. 12, bottom row). The LSTM-AE features outperform the expert indicators in all cases. Therefore, the LSTM-AE is able to produce features that are better than the expert system at preserving the information of the original cycles.

Discussion

In a ranking evaluation, the LSTM-AE features outperform the expert indicators. Future works will include conducting more robust evaluations, having more than 40 features, and adding more contextual variables other than the train stations.

Nevertheless, we discover a fundamental issue in the underlying principle of autoencoders: they erase input perturbations in favor of a smoother reconstruction. Because autoencoders strive to learn the most representative abstraction of the inputs, they are insensitive to input disturbances (e.g., a sudden peak in the signals). However, such noises could be indicative of a system anomaly. Moreover, the limited interpretability of the LSTM-AE features is an issue of equal importance. Possible improvements include (i) adding explainability to the LSTM-AE to interpret the magnitude of change in the features with respect to the perturbations in the cycles, (ii) customizing the loss function and/or the network architecture to retain relevant perturbations, and (iii) adjusting the LSTM-AE architecture to further improve the feature quality.

6 Health detection

Each feature vector represents a cycle and is used to analyze the condition of the systems. To predict the failures in a system, we must first identify its current condition, or *health*. From the literal definition of "health", we define the health of a system as *the extent to which it is free of anomaly*, quantified by a score in the interval [0, 1], with 1 representing the worst health (full of anomalies) and 0 representing the best health (free of anomaly). Because the railway operating constraint requires that the majority of the fleet be operational at all times, the majority of data from the fleet defines normal health. A system is afflicted with an anomaly if it exhibits behavior that deviates from what is expected. The

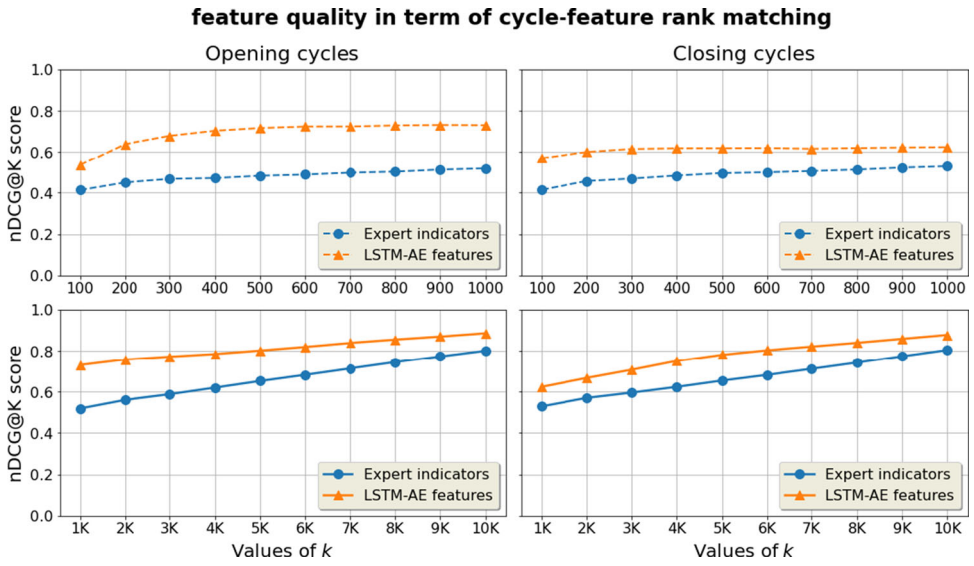


Fig. 12 Cycle-feature ranking quality via nDCG@k

health score of a system is calculated based on the cumulative effect of the anomalies affecting it and the severity of each anomaly.

We aim to identify the *health profiles* of the fleet from the data, such that one health profile is constructed from the data of systems that in under the same health condition, for example, one health profile for PASs that fail to open and one for PASs that close more slowly than expected. A health profile can be normal (representing good health) or anomalous (representing degradation). Because the systems are designed to function in a unique normal state, we have a unique normal health profile, also known as the *reference profile*, as well as multiple anomaly profiles (Fig. 13).

6.1 CheMoc

To determine the current health of the systems from the stream of feature vectors, we propose Continuous Health Monitoring using Online Clustering, or CheMoc, to discover the collective health profiles of the fleet, upon which the health of each system is computed individually and adaptively. In the following, we refer to "feature vectors" and "data points" interchangeably.

6.1.1 Fundamental concepts

Let $D(T) = \{ X^t_{S_m} \mid S_m \in \mathcal{S}, 1 \leqslant t \leqslant T \}$ be a stream of feature vectors from the beginning (when the first cycle arrives) until T. From $D(T)$, a set $\mathcal{G}(T) = \{ G_1(T), \ldots, G_K(T) \}$ of K health profiles can be discovered and updated incrementally, which is why $\mathcal{G}(T)$ and $G_k(T) \in \mathcal{G}(T)$ are functions of time. The reference profile is denoted $\overline{G}(T)$ and is included in $\mathcal{G}(T)$.

Each health profile has a severity $\omega_k(T) \in [0, 1]$, with higher values indicating more severe anomalies. The reference $\overline{G}(T)$ always has $\overline{\omega}(T) = 0$. The severity of an anomaly profile depends on how different it is from $\overline{G}(T)$. To quantify the difference, we store in $G_k(T)$ (and $\overline{G}(T)$) a *feature matrix* $\mathbf{F}_k$ ($\overline{\mathbf{F}}(T)$) that incrementally updates the statistics of the data points falling in this profile. The deviation $\Delta_k(T)$ of a profile $G_k(T)$ from $\overline{G}(T)$ is the Frobenius of $\mathbf{F}_k(T)$ and $\overline{\mathbf{F}}(T)$ (9). We divide $\Delta_k(T)$ by the maximum $\Delta_{k'}(T)_{1 \leqslant k' \leqslant K}$ to obtain the severity $\omega_k(T)$ bounded in [0, 1] (10).

$$\Delta_k(T) = \left\| \mathbf{F}_k(T) - \overline{\mathbf{F}}(T) \right\|_F > 0 \tag{9}$$

$$\omega_k(T) = \frac{\Delta_k(T)}{\max\limits_{k'} \Delta_{k'}(T)} \tag{10}$$

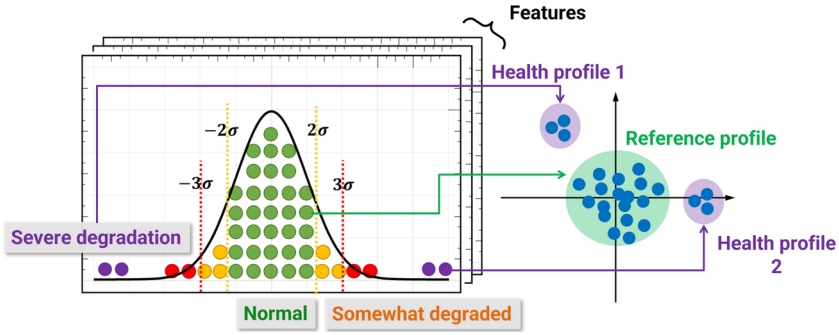


Fig. 13 Examples of health profiles as clusters captured from multiple indicators [36]

From the health profiles and their severity, we quantify the degree to which an anomaly $G_k(T)$ manifests in a system S_m at T via an anomaly score $A_k^{S_m}$. Intuitively, the more data S_m has in an anomaly profile, the more severe this anomaly is affecting S_m. Let $G_k^{S_m}(T)$ be the set of feature vectors from a system S_m in G_k at T, and $\overline{G}^{S_m}(T)$ for $\overline{G}(T)$, the anomaly score is obtained by:

$$A_k^{S_m}(T) = \frac{|G_k^{S_m}(T)|}{|G_k^{S_m}(T)| + |\overline{G}^{S_m}(T)|} \in [0, 1] \tag{11}$$

However, (11) disregards the recency of the data and only considers the number of data points produced by a system in a cluster up until T. Meanwhile, old data on a data stream become less relevant over time. To account for this temporal decay, we add a decay factor $f(t) = 2^{-\lambda t}$ ($\lambda > 0$, higher λ means faster decay) to each feature vector in $G_k^{S_m}(T)$ and $\overline{G}_k^{S_m}(T)$. Adjusting λ is application-dependent. The anomaly score $A_k^{S_m}(T)$ thus becomes:

$$A_k^{S_m}(T) = \frac{\sum_{i}^{G_k^{S_m}(T)} f(T_m - T_i)}{\sum_{i}^{G_k^{S_m}(T)} f(T_m - T_i) + \sum_{j}^{\overline{G}_k^{S_m}(T)} f(T_m - T_j)} \tag{12}$$

where T_m is the timestamp of the last cycle created by S_m on $D(T)$, T_i and T_j the creation time of each cycle of S_m in $G_k^{S_m}(T)$ and $\overline{G}_k^{S_m}(T)$, respectively.

The health score $H_{S_m}(T)$ is the average of all the anomaly scores of S_m at T, weighted by the severity of each anomaly, so that a low anomaly score in a severe anomaly profile still casts a large impact on the health score (13). We omit the reference profile from the health score formula because we only consider the anomalies manifesting in the system.

$$H_{S_m}(T) = \frac{1}{|\mathcal{G}(T) \setminus \overline{G}(T)|} \sum_{k \in \mathcal{G}(T) \setminus \overline{G}(T)} \omega_k(T) A_k^{S_m}(T) \tag{13}$$

6.1.2 Learning the health profiles with online clustering

From the fundamental concepts, we seek a suitable online clustering algorithm that enables us to discover the evolving health profiles of the systems. Because we lack a ground truth for the PASs' health profiles, the first phase is to obtain a baseline using a simple, easy-to-manage online clustering algorithm. Once the baseline has been validated, we can experiment with other clustering algorithms for improvements.

Among existing online clustering algorithms,[7] such as DenStream [6], D-Stream [9], ESA-Stream [21], ClusTree [18], FlockStream [13], we implement DenStream as the core clustering of CheMoc for the following reasons.

First, DenStream is a density-based algorithm that does not need a predefined number of clusters, making it suitable for online scenarios where the exact number of anomalies is unknown in advance. Secondly, DenStream makes use of a lightweight data structure that is efficiently and incrementally updatable on fast data streams. Thirdly, although new algorithms have been introduced since the publication of DenStream in 2006, DenStream remains a foundation for numerous online algorithms (e.g., SDSStream [27], rDenStream [23], C-DenStream [29]).

We will begin by describing the underlying principle of DenStream and how we adapt it to make it compatible with the specific operational constraints of the railway.

DenStream DenStream is a two-phase clustering algorithm. During the online phase, DenStream captures statistics from the stream in the form of lightweight *micro-clusters*. During the offline phase, DenStream converts the centroid of these micro-clusters to virtual points, on which it performs a traditional clustering using DBSCAN [11] to return the official clusters.

A micro-cluster mc_i at a time t for a group of n points $p_{i_1}, \ldots, p_{i_n}$ created at timestamps $T_{i_1}, \ldots, T_{i_n}$ is defined by a tuple of cluster features $\{w, LS, SS\}$ in which w is the weight (14), LS is the linear sum (15), and SS is the squared sum (16). Then, we can easily compute its centroid $\overline{c}$ (17) and radius r (18) ($|.|$ denotes the L1 norm). The radius of mc_i must be lower than ϵ which defines the desired density of any micro-clusters. The function $f(t) = 2^{-\lambda t}(\lambda > 0)$ is the exponential decay function that dictates the importance of a data point at a time t. Higher λ means faster forgetting.

$$w = \sum_{j=1}^{n} f(t - T_{i_j}) \tag{14}$$

$$LS = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j} \tag{15}$$

$$SS = \sum_{j=1}^{n} f(t - T_{i_j}) (p_{i_j})^2 \tag{16}$$

$$\overline{c} = \frac{LS}{W} \tag{17}$$

$$r = \sqrt{\frac{|SS|}{W} - \left(\frac{|LS|}{W}\right)^2} \quad < \epsilon \tag{18}$$

[7] We refer to these excellent surveys for more details on online clustering algorithms [7, 39].

A micro-cluster mc_i is updated when it receives a new point p at t (19), or when it does not receive any new points after an interval Δt and decays (20). In both cases, the cluster features can be incrementally updated using simple operations, which make micro-clusters efficient to maintain on a data stream.

$$mc_i = \{ w + 1,\ LS + p,\ SS + p^2 \} \tag{19}$$

$$mc_i = \{ 2^{-\lambda \Delta t} w,\ 2^{-\lambda \Delta t} LS,\ 2^{-\lambda \Delta t} SS \} \tag{20}$$

A micro-cluster can be a potential micro-cluster (PMC) or an outlier micro-cluster (OMC) depending on its weight. Let μ be the minimum weight of a dense micro-cluster and $0 < \beta \leqslant 1$ the outlierness threshold, a PMC must satisfy $w \geqslant \beta\mu$, or else it is an OMC. Only the PMCs become virtual points for offline clustering.

On a data stream, a PMC degrades to an OMC if it does not receive data for an extended period of time, or an OMC evolves into a PMC if it receives sufficient data to become dense. An OMC that does not receive any new data is considered a noise and will be discarded.

DenStream performs periodic pruning every interval $T_p = \left\lceil \frac{1}{\lambda} \log(\frac{\beta}{\beta\mu - 1}) \right\rceil$ to decay the micro-clusters, then demotes PMCs and discards OMCs whose weight is below the acceptable threshold.

The weight threshold of a PMC is $\beta\mu$, and of an OMC is $\xi(t, t_o) = \frac{2^{-\lambda(t - t_o + T_p)} - 1}{2^{-\lambda T_p} - 1}$, which is as a function of the OMC's creation time t_o and of the current time t.

Adapting DenStream to CheMoc The original implementation of DenStream is incompatible with the railway scenario. We will describe how we modified DenStream to correct the misalignment.

(a) Offline clustering omitted

Specifically, DenStream has two offline procedures. One is a warm-start, which initializes DenStream on a limited amount of data to produce the initial micro-clusters. The second performs a standard DBSCAN clustering on PMCs as virtual points. Experiments have shown that the warm-start process stabilizes the growth of clusters in CheMoc. Without it, DenStream tends to generate an excessive number of OMCs and is unable to identify meaningful micro-clusters. As the warm-start uses a small amount of data and completes rapidly, it is not a bottleneck in DenStream.

We omit the second offline process. DenStream employs offline clustering to eliminate outdated OMCs in the hopes of finding a noise-free set of clusters. In the railway, a system can only be in a finite number of health profiles and a system changes from one profile to another gradually; therefore, abrupt changes in the data distribution are uncommon. Even if a sudden change occurs, the resulting effect will be attenuated by the data coming from other systems in $\mathcal{S}$, resulting in stable micro-clusters during the online phase.

During the offline clustering phase, each micro-cluster is transformed into a single virtual point representing its centroid. As a result, this offline clustering phase is not only unnecessary, as previously explained, but it can also be detrimental because a meaningful micro-cluster representing a health profile is collapsed into a single virtual point, which causes information loss. In addition, the clusters obtained between each offline clustering are disconnected and their temporal evolution is difficult to observe. This disrupts the inherent continuity of evolving micro-clusters.

Considering these factors, we only maintain the warm-start and the online phase of DenStream. From now on, we regard a micro-cluster to be an official cluster.

(b) Dynamic density threshold

In DenStream, ϵ defines the desired density of the clusters (18). Nevertheless, it is difficult to guess the true density of a data stream. A high ϵ to find clusters on sparse data can lead to excessively large clusters that mistakenly group data from different health profiles when the data become denser. Ideally, ϵ should be dynamically adjusted based on the current density of the stream.

To do so, we implement the adaptive scheme proposed by Putina and Rossi [26], by revising ϵ continuously based on the radius of the clusters. Let r be a random variable that records the radii of all clusters maintained thus far, $\bar{r}$ and σ_r the mean and the standard deviation of r computed incrementally, the new density is set to:

$$\epsilon = \bar{r} + k\sigma_r \qquad \text{with} \qquad k > 0 \tag{21}$$

We set $k = 3$ as recommended by Putina and Rossi [26].

(c) Pruning omitted

DenStream periodically prunes PMCs and OMCs that have not been updated recently in order to eliminate obsolete clusters. However, clusters that depict health profiles should not be discarded. A cluster that has not received data for some time is not inherently outdated, but rather because no system fits that profile during a particular time period. This commonly happens in the railway. If we discard inactive clusters, CheMoc loses knowledge of these health profiles and must relearn them if these profiles are activated in the future, resulting in suboptimal performance. Therefore, we remove the periodic pruning from DenStream. As a reminder, DenStream only decays the clusters' weight during the pruning phase. Omitting the periodic pruning implies that the clusters will not decay over time.

Yet, we must track the temporal evolution of the system's health. Although the clusters do not lose their relevance, the system's data do. The data produced by a system long ago

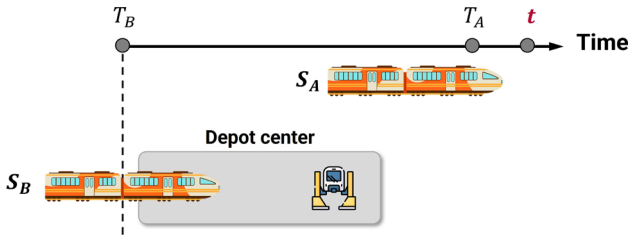


Fig. 14 The system S_B enters a depot center at T_B and stops producing data, whereas the system S_A continues to operate until T_A. The recency of S_B's data is not dependent on t (the current timestamp), but rather on T_B (the last time S_B produced data)

no longer reflect its current health (for example, the system was healthy a few weeks ago but has since degraded). This means the decay must occur *on the data*, such that older data contribute less to the anomaly scores than more recent data. This is effectively captured by (12) that incorporates the decay factor into the computation and lowers the influence of old data over time, thereby ensuring the adaptability of CheMoc.

(d) Varying anchor timestamp by system

The weight of a cluster mc_i of n data points is $w = \sum_{j=1}^{n} f(t - T_{i_j})$, where T_{i_j} is the timestamp of the point p_{i_j} in mc_i and t the current time. Normally, t applies to all points in mc_i such that a point p_{i_j} is less relevant the farther its timestamp T_{i_j} is to t.

However, a system that does not produce new data does not necessarily have less relevant data than one that does. It is possible that the former is at rest and stops producing data, while the latter continues to operate (Fig. 14). The recency of the data of a system is not determined by a global timestamp, but by the timestamp of the most recent data point of each system. Therefore, a global timestamp t cannot be used to decay the data of different systems within a cluster; instead, t must vary by system. This results in a new definition of a cluster mc_i.

As a reminder, $\mathcal{S} = \{ S_1, ..., S_M \}$ is the set of M systems being monitored. Let $\psi(p)$ be a function that maps a point p to the system that produces it.

For a cluster mc_i, we define $\mathcal{S}_i$ as a set of M_i systems such that each system $S_{i_m} \in \mathcal{S}_i$ produces at least one data point in mc_i ($S_{i_m} \in \mathcal{S}_i$, $M_i \leqslant M$, $\mathcal{S}_i \subseteq \mathcal{S}$) (22).

We denote $\mathcal{P}_{i_m}$ as the set of n_{i_m} data points produced by a system S_{i_m} in $\mathcal{S}_i$ (23).

$$\mathcal{S}_i = \{ S_m \in \mathcal{S} \mid \exists p \in mc_i,\ \psi(p) = S_m \} \tag{22}$$

$$\mathcal{P}_{i_m} = \{ p \in mc_i \mid \psi(p) = S_{i_m} \} \tag{23}$$

Let $mc_{i_m} = \{ w_{i_m}, LS_{i_m}, SS_{i_m} \}$ be the tuple of cluster features computed on the data points produced by each system $S_{i_m} \in \mathcal{S}_i$ in mc_i such that w_{i_m} is the weight by S_{i_m} (24), LS_{i_m} the linear sum by S_{i_m} (25), SS_{i_m} the squared sum by S_{i_m} (26), and t_m the timestamp of the latest point produced by S_{i_m} on the stream. Because the latest timestamp t_m of a system S_m is not local to any cluster, the index i is omitted.

$$w_{i_m} = \sum_{j=1}^{n_{i_m}} f(t_m - T_{i_m}^j) \tag{24}$$

$$LS_{i_m} = \sum_{j=1}^{n_{i_m}} f(t_m - T_{i_m}^j)\, p_{i_m}^j \tag{25}$$

$$SS_{i_m} = \sum_{j=1}^{n_{i_m}} f(t_m - T_{i_m}^j)(p_{i_m}^j)^2 \tag{26}$$

The cluster features of mc_i is the sum of all tuples by system in $\mathcal{S}_i$ (27).

$$mc_i = \{ w, LS, SS \} = \left\{ \sum_{S_{i_m}}^{\mathcal{S}_i} w_{i_m}, \sum_{S_{i_m}}^{\mathcal{S}_i} LS_{i_m}, \sum_{S_{i_m}}^{\mathcal{S}_i} SS_{i_m} \right\} \tag{27}$$

The new definition still enables incremental cluster updates. Let us consider two cases: (i) mc_i receives a new point p and (ii) mc_i decays.

If p is added to mc_i, p may or may not come from a system that already has data in mc_i. We distinguish two sub-cases, where $\psi(p) \in \mathcal{S}_i$ and $\psi(p) \notin \mathcal{S}_i$.

- If $\psi(p) = S_{i_m} \in \mathcal{S}_i$, the latest timestamp t_m of S_{i_m} is updated to T when S_{i_m} produced p at T. The decay factor when p is merged in mc_i becomes $f(t_m - T) = f(0) = 1$, and mc_{i_m} thus becomes:

  $$\{ w_{i_m} + 1,\ LS_{i_m} + p,\ SS_{i_m} + p^2 \}$$

  Following (27), mc_i after merging p is:

  $$mc_i = \{ w + 1,\ LS + p,\ SS + p^2 \} \tag{28}$$

- If $\psi(p) = S_l \notin \mathcal{S}$, the cluster features of S_l is:

  $$mc_{i_l} = \{ w_{i_l},\ LS_{i_l},\ SS_{i_l} \} = \{ 1,\ p,\ p^2 \}$$

  Following (27), mc_i after merging p (and S_l) is:

  $$mc_i = \{ w + 1, LS + p, SS + p^2 \} \tag{29}$$

For the second case, decaying a cluster means decaying its cluster features. For each system $S_{i_m} \in \mathcal{S}_i$, let $\Delta t_{i_m} = t_m - T_{i_m}$ be the time interval from the latest point by S_{i_m} on the stream (t_m) to the latest point of S_{i_m} in a specific cluster mc_i (T_{i_m}). To decay mc_i, we sum the decayed cluster features

of each system in mc_i. For example, the decayed linear sum LS' of mc_i becomes:

$$
\begin{aligned}
LS' &= \sum_{S_{i_m}}^{\mathcal{S}_i} LS'_{i_m} = \sum_{S_{i_m}}^{\mathcal{S}_i} \sum_{j=1}^{n_{i_m}} f(\Delta t_m + t_m - T_{i_m}^j) p_{i_m}^j \\
&= \sum_{S_{i_m}}^{\mathcal{S}_i} \sum_{j=1}^{n_{i_m}} 2^{-\lambda(\Delta t_m + t_m - T_{i_m}^j)} p_{i_m}^j \\
&= \sum_{S_{i_m}}^{\mathcal{S}_i} \sum_{j=1}^{n_{i_m}} 2^{-\lambda \Delta t_m} 2^{-\lambda(t_m - T_{i_m}^j)} p_{i_m}^j \\
&= \sum_{S_{i_m}}^{\mathcal{S}_i} 2^{-\lambda \Delta t_m} \sum_{j=1}^{n_{i_m}} 2^{-\lambda(t_m - T_{i_m}^j)} p_{i_m}^j \\
&= \sum_{S_{i_m}}^{\mathcal{S}_i} 2^{-\lambda \Delta t_m} LS_{i_m}
\end{aligned}
$$

SS' and w' are decayed similarly. Therefore, it suffices to multiply the features $\{ w_{i_m}, LS_{i_m}, SS_{i_m} \}$ of each system $S_{i_m} \in \mathcal{S}_i$ to $\eta = 2^{-\lambda \Delta t_m}$ and sum the decayed features of all the systems in mc_i to decay mc_i incrementally (30).

$$
mc_i = \left\{ \sum_{S_{i_m}}^{\mathcal{S}_i} \eta w_{i_m}, \sum_{S_{i_m}}^{\mathcal{S}_i} \eta LS_{i_m}, \sum_{S_{i_m}}^{\mathcal{S}_i} \eta SS_{i_m} \right\} \tag{30}
$$

In conclusion, the new definition of cluster features by system still enables on-the-fly cluster updates via (28), (29), and (30).

Full algorithm Algorithm 12 describes CheMoc step-by-step. W is the buffer size to warm-start CheMoc, M is the online clustering algorithm (DenStream in our case), and θ is the hyperparameters of the clustering algorithm that vary depending on M.

Algorithm 4 CheMoc.

Data: A stream of data points $D(T)$
Input: Buffer size W, clustering hyperparameters θ

```
1  M.init(θ)                          /* init DenStream */
2  WS ← ∅                    /* init warm start buffer */
3  while stream D(T) is active do
4      x ← new data point from D(T)
5      if not yet warm started then          /* warm start */
6          if |WS| = W then M.warmStart(WS)
7          else WS ← WS ∪ {x}
8      else
9          M.process(x)                  /* update G(T) */
10         if is update time after an interval T_u then
11             M.reassessClusters()
12             M.updateHealth()
```

First, M is initialized on θ. An empty buffer WS is allocated to store the data for warm-starting CheMoc. The warm-start follows the procedure described in [6]. After the warm-start (line 5-7), CheMoc updates the clusters on each new data point (line 9). At every interval T_u,[8] CheMoc reassesses the clusters and recomputes the health score of the systems. During the reassessment, the cluster with the most data points is elected to be the reference profile $\overline{G}(T)$ (line 11). Once $\overline{G}(T)$ is identified, the severity $\omega_k(T)$ of each cluster $G_k(T)$ is recomputed using (9) and (10). Then, the health score of each system is updated using (12) and (13) (line 12).

6.2 Experimental results

We test CheMoc on 1,074,272 feature vectors. To simulate a data stream, we input the data points to CheMoc sequentially in chronological order. Instead of performing the health update for each data point (line 10-12 in Algorithm 12), we use a *micro-batch* approach: each time CheMoc receives a batch of one-week data, it updates the clusters on every data point in this batch, reassesses the clusters, and updates the health scores. Since we log the changes within CheMoc in a database for post-analysis, micro-batching reduces the bottleneck caused by database operations.

6.2.1 Experimental setup

We use the number of cycles as the time unit in CheMoc because, in the railway industry, the number of cycles is more significant than time expressed in seconds or hours. A system only degrades when it is in operation, during which it continuously produces new cycles. An inactive system, on the other hand, generates no data and does not degrade over time. Therefore, in CheMoc, all notions of time are expressed in the number of cycles.

Because CheMoc uses DenStream as its core clustering, four important hyperparameters are the desired density ϵ, the decay factor λ, the outlierness threshold β, and the minimum weight μ of a dense cluster (Table 3).

To select the initial value of ϵ, we compute the mean and standard deviation of the pairwise distances on a random data sample and set $\epsilon_0 = \lceil \mu_{dist} + \sigma_{dist} \rceil$. This value of ϵ is continuously adjusted on the data stream (21).

The decay factor λ is computed such that a cycle is forgotten after a window of 100 next cycles, as suggested by a

[8] This can be carried out for every new data point or by micro-batch. Updating on every point is compatible with the principle of real-time monitoring, but it can cause data communication bottleneck if the traces must be saved in a database. This choice is therefore application-dependent.

Table 3 Hyperparameters of CheMoc

Parameters	Value	Meaning
ϵ	15	The initial density for the warm-start
λ	0.06	Forgetting a cycle after the 100 next cycles
β	0.4	Outlierness threshold of a cluster
μ	5	Minimum number of points per system in a cluster

domain expert. We set this window to be $\Delta t = 100$ and rule that a cycle is forgotten if its relevance drops to 0.01, i.e., $2^{-\lambda \Delta t} = 0.01$. Hence, we have $2^{-\lambda \Delta t} = 2^{-100\lambda} = 0.01 \Rightarrow \lambda \approx 0.06$.

The outlierness threshold β determines if a cluster is dense enough to become a PMC ($\beta\mu$). The higher the value of β, the denser a cluster must become to be considered a PMC. We pick the value 0.4 to be on the tolerant side when promoting an OMC to a PMC.

We modify the semantics of the minimum weight μ such that it imposes the least number of points a system must have in a cluster to be allowed to update the cluster's features. The aim is to eliminate noises from systems that have very few data points in a cluster.

The data we stored in the statistic matrices $\mathbf{F}_k(T)$ and $\overline{\mathbf{F}}(T)$ are the mean of all data in a cluster, which is simply its centroid. CheMoc is warm-started on the first batch of data with 1589 data points. The warm-start finishes after three seconds.

6.2.2 Cluster analysis

In total, CheMoc captures 12 clusters, including 11 PMCs ($\overline{G}$, G_1-G_{10}) and one OMC (G_{11}). We project the full data on a 3D space using Principle Component Analysis. The data exist in a large cloud with significant outliers scattering on the border, but they do not exhibit intricate topology (Fig. 15a).

CheMoc discovers a set of tight-knit clusters (Fig. 15c). The cluster distribution implies that when a system degrades, its health deviates gradually from the reference profile but does not become completely separated. A small ϵ is therefore better to detect such gradual trajectory of a system's data via small clusters that change their position over time. The temporal trajectory of the clusters confirms that the clusters stabilize after the first few batches and move slowly afterward (Fig. 15b).

Then, dynamically adjusting ϵ makes it increasingly smaller (Fig. 16b), which means the clusters become denser over time. Their radii concentrate around 0.7 to 1.5 (Fig. 16a). The data have a stable structure and CheMoc has converged to a set of known health profiles, positively confirmed by a domain expert.

To track the evolving health of the systems, we randomly pick one system (denoted S_m) to analyze its health evolution. Figure 17 plots the health score $H_{S_m}(T)$ updated at each batch against the amount of data by S_m in each cluster. Because S_m mostly produces data in anomaly profiles, $H_{S_m}(T)$ is relatively high and rises as S_m produces more data in anomalous clusters.

To examine the anomalies that impacted S_m, we select one batch and visualize the data in the clusters that contribute to H_{S_m}. We choose the batch just before 03/05/2021, when H_{S_m} starts rising. In this batch, S_m appears in multiple clusters, so we visualize the most notable ones, namely orangeG_1, G_3, G_4, and G_6 (Fig. 18). We compute the profile (thick line) and envelope (colored region) of these clusters based on the cycles they contain, using the three most relevant measurements: the current, the voltage, and the position of the electric motor in the PAS. We plot both the profile of a cluster G_k and that of the reference cluster $\overline{G}$ for comparison.

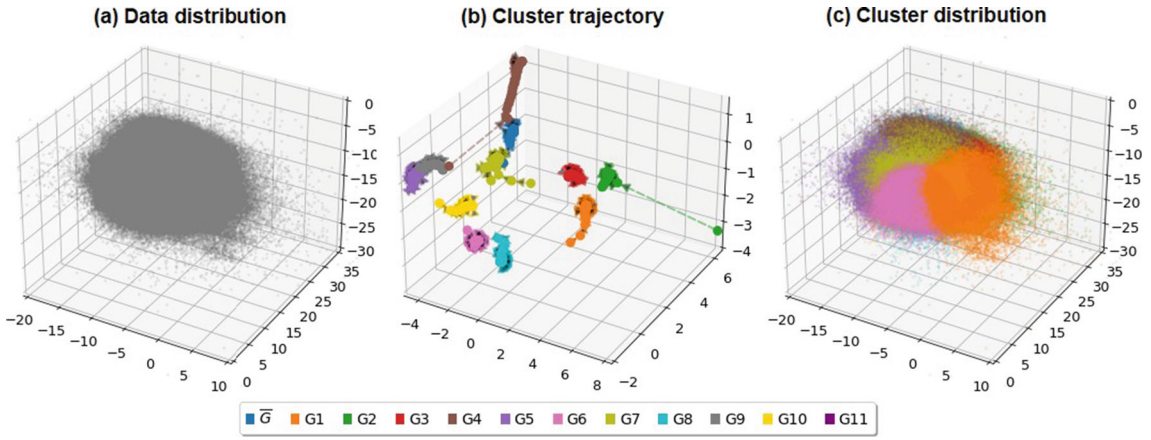


Fig. 15 Visualization of one-year data (a), of the cluster trajectory (b), and of the final clusters (c)

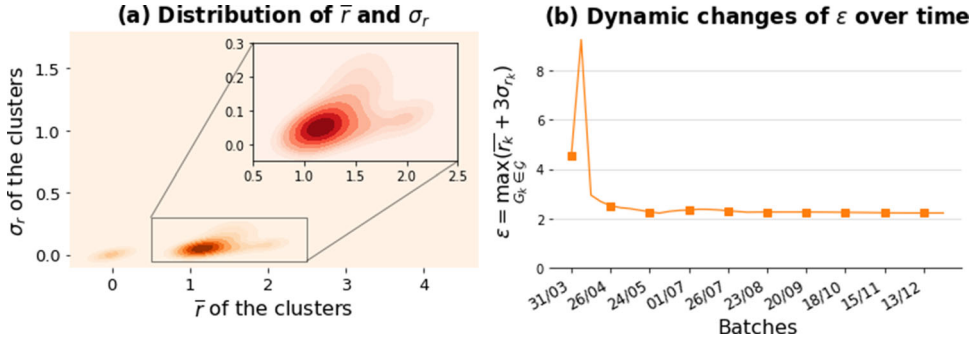


Fig. 16 Dynamically adjusted ϵ via the mean $\overline{r}_k$ and standard deviation σ_{r_k} of the clusters' radii

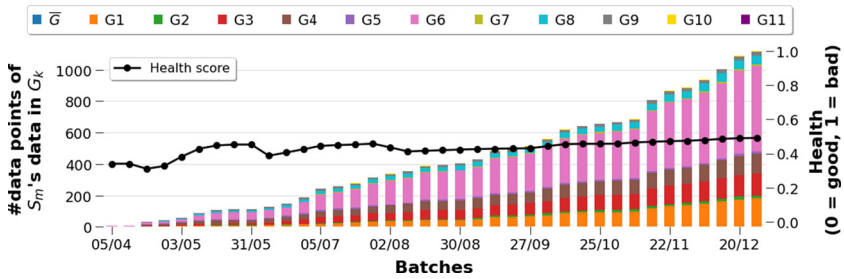


Fig. 17 Health score of S_m versus the amount of data from S_m in the clusters

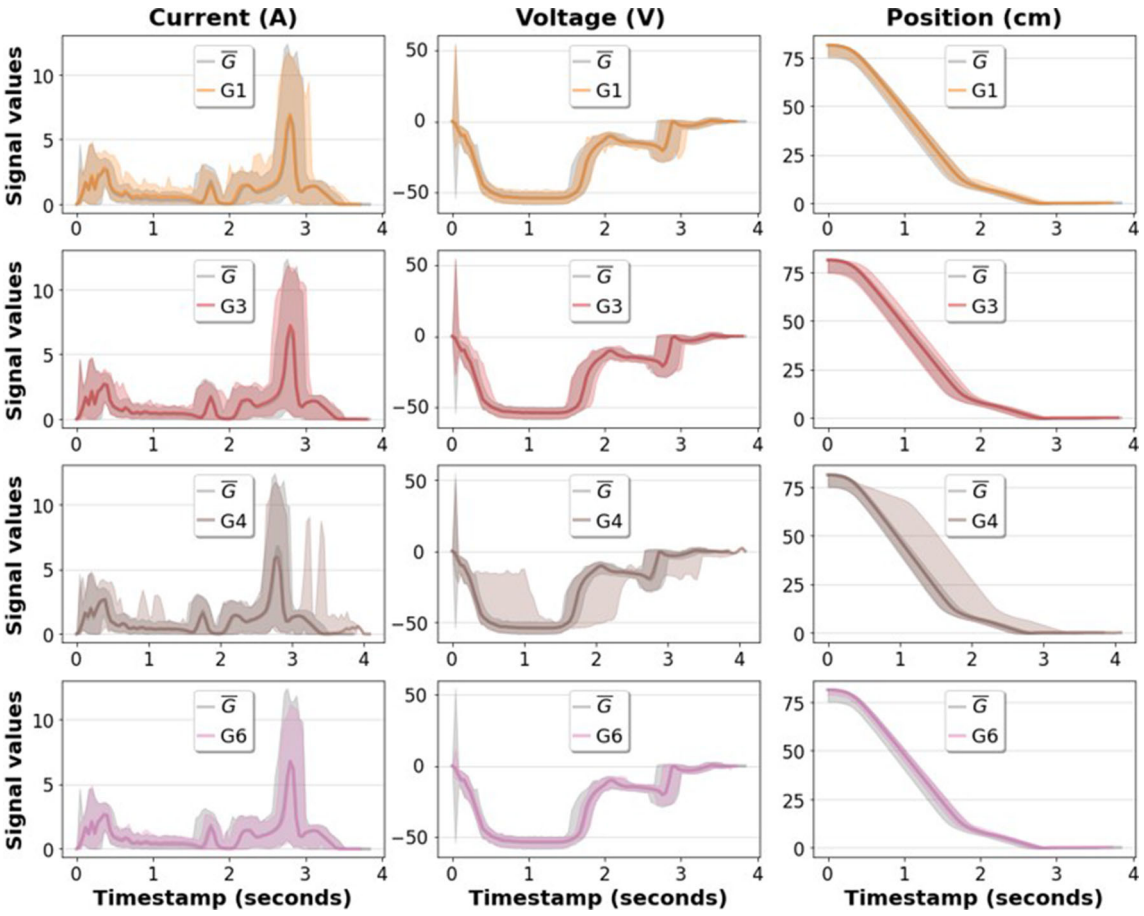


Fig. 18 The profiles of G_1, G_3, G_4, and G_6, via the current, voltage, and position measurements

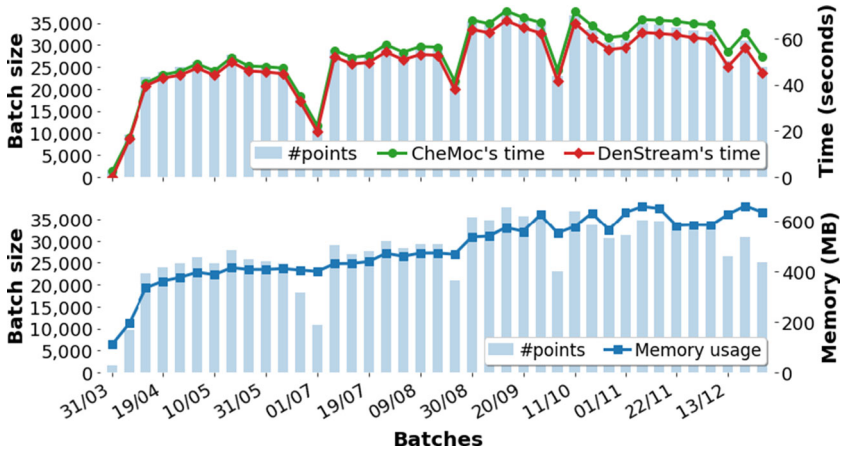


Fig. 19 Processing time and memory usage of CheMoc

Figure 18 shows that G_1, G_3, G_4, and G_4 display behaviors different from that of the normal profile $\overline{G}$ (light gray), especially G_4. The presence of S_m's data in these clusters means that S_m deviated from good health, which in turn increases its health score.

Besides the accuracy of cluster discovery and health assessment, monitoring a significant number of systems continuously requires efficiency. Figure 19 depicts the processing time and memory usage of CheMoc on each batch with respect to the batch size. The execution time of CheMoc is dominated by that of DenStream, with a minor gap for cluster reassessment and health score recomputation. The warm-start for the first batch is completed in three seconds. Overall, the processing time is proportional to the batch size.

The memory usage of CheMoc increases slowly. When data from a new system arrive, CheMoc must store the cluster features of the new system in the clusters, resulting in a gradual increase in memory consumption. Nonetheless, processing one year of data from an entire fleet requires only 600 MB. As the number of systems is finite, memory consumption will not increase indefinitely.

Discussion

CheMoc is capable of capturing health profiles by clustering an unlabeled data stream and estimating the health scores of monitored systems while remaining computationally efficient. However, it has two significant flaws.

Currently, CheMoc maintains the relevance weights of all data points in memory in order to rapidly compute the anomaly scores (12). Although only the weight of each point rather than its entire content is stored, this solution is not scalable on an unbounded data stream. To avoid storing individual relevance weights, we can transform (12) as follows.

Let $dG_k^{S_m} = \sum_i^{|G_k^{S_m}(T)|} f(t_m - T_i)$ be the sum of the relevance weights of all the points from S_m in $G_k(T)$, where t_m is the latest timestamp of S_m on $D(T)$ and T_i is the timestamp of the point p_i of S_m in $G_k(T)$.

$$\begin{aligned} dG_k^{S_m} &= \sum_i^{G_k^{S_m}(T)} f(t_m - T_i) = \sum_i^{G_k^{S_m}(T)} 2^{-\lambda(t_m - T_i)} \\ &= \sum_i^{G_k^{S_m}(T)} 2^{-\lambda t_m} 2^{\lambda T_i} \\ &= 2^{-\lambda T_c} \sum_i^{G_k^{S_m}(T)} 2^{\lambda T_i} \end{aligned} \tag{31}$$

The term $2^{\lambda T_i}$ in (31) can be computed incrementally by cumulatively summing $2^{\lambda T_i}$ for every new point p_i from S_m at T_i in $G_k(T)$, eliminating the need of storing individual relevance weights. However, computing $2^{\lambda T_i}$ becomes numerically infeasible as $T_i \to \infty$. To amend this, we can introduce a "landmark" timestamp T_o:

$$dG_k^{S_m} = \sum_i^{G_k^{S_m}} f(t_m - T_o - T_i + T_o) = 2^{-\lambda(t_m - T_o)} \sum_i^{G_k^{S_m}} 2^{\lambda(T_i - T_o)}$$

such that $T_i - T_o$ should be much smaller than T_i. Moreover, T_o brings an additional subtlety: all the data recorded before T_o are considered obsolete and are ignored when we calculate (12). For instance, we can set T_o to be the moment of the last maintenance that restored a system. Then, $T_i - T_o$ becomes the interval from the *last-known time of good health* of a system until T_i. Without an explicit T_o, T_i implies that a system's last-known time of good health is at 0, which is the start of its lifecycle. Yet, T_o is difficult to obtain, because maintenance feedback is not always accessible in practice.

The second problem is that CheMoc lacks a feedback loop. The resulting clusters are as good as they could be

without any input from experts. When evaluating CheMoc, we gather batch-by-batch cluster footprints and present them to a domain expert. They validate the clusters using their knowledge of the systems. We make adjustments to CheMoc based on their feedback in an effort to discover better clusters. This procedure is repeated for every experiment.

We recognize that it is preferable to incorporate the feedback directly into CheMoc, forming a feedback loop that enables CheMoc to improve itself, similar to what we did for InterCE. A feedback may prompt CheMoc to re-evaluate its hyperparameters, divide or merge clusters. Finally, the goal of CheMoc is not to replace domain experts, but to serve as a tool to aid them in their decision-making process.

7 Conclusion

To address predictive maintenance of railway systems on real-time data streams, we devise three methods: InterCE, LSTM-AE, and CheMoc, which address automatic cycle extraction, learning features, and machinery health monitoring. Our methods are evaluated using real-world data provided by SNCF, the French national railway company.

InterCE surpasses the baseline extraction accuracy of the expert system, achieves a nearly constant processing time per input (approximately 1.5s), and has a low query ratio (0.04% over 100,000 inputs).

The LSTM-AE generates features that are more compact and better at conserving information than the indicators identified by the expert system. Nevertheless, the LSTM-AE tends to attenuate minor perturbations that may be associated with system degradation. This issue requires further investigation.

CheMoc is capable of discovering a set of relevant health profiles of the systems using the collective data from a fleet and adaptively assessing the system's health with satisfactory efficiency.

These methods demonstrate the potential of online machine learning for predictive maintenance and showcase the benefits of continuous, lifelong learning over static offline learning. The proposed methods will be improved in future studies.

Acknowledgements We are grateful to the SNCF for providing us with the data used to develop and evaluate our methods.

Author Contributions All authors contribute to the manuscript equally.

Funding Funded by the *Association Nationale de la Recherche et de la Technologie (ANRT) de la France*.

Data Availability Due to the confidential obligation imposed by the data supplier, we cannot publish our data.

Code Availability The source code implementing the methods described in this article is available at https://tinyurl.com/ms4bvj5k.

Declarations

Competing interests We declare that there is no competing interests among the authors.

Author approbation All authors have approved the manuscript for submission.

References

1. Abadi M, Agarwal A, Barham P et al (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, software available from tensorflow.org
2. Aminikhanghahi S, Cook DJ (2017) A survey of methods for time series change point detection. Knowl Inf Syst 51(2):339–367. https://doi.org/10.1007/s10115-016-0987-z
3. Aydemir G, Acar B (2020) Anomaly monitoring improves remaining useful life estimation of industrial machinery. J Manuf Syst 56:463–469
4. Ben Ali J, Saidi L, Harrath S et al (2018) Online automatic diagnosis of wind turbine bearings progressive degradations under real experimental conditions based on unsupervised machine learning. Appl Acoust 132:167–181. https://doi.org/10.1016/j.apacoust.2017.11.021
5. Canizo M, Onieva E, Conde A et al (2017) Real-time predictive maintenance for wind turbines using big data frameworks. In: 2017 IEEE international conference on prognostics and health management (ICPHM). pp 70–77. https://doi.org/10.1109/ICPHM.2017.7998308
6. Cao F, Ester M, Qian W et al (2006) Density-based clustering over an evolving data stream with noise. In: Proceedings of the 6th SIAM international conference on data mining, April 20-22, 2006, Bethesda, MD, USA, https://doi.org/10.1137/1.9781611972764.29
7. Carnein M, Trautmann H (2019) Optimizing data stream representation: An extensive survey on stream clustering algorithms. Business & Information Systems Engineering 61(3):277–297. https://doi.org/10.1007/s12599-019-00576-5
8. Carvalho TP, Soares FAAMN, Vita R et al (2019) A systematic literature review of machine learning methods applied to predictive maintenance. Computers & Industrial Engineering 137:106024
9. Chen Y, Tu L (2007) Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. Association for Computing Machinery, San Jose, California, USA, KDD '07. pp 133–142. https://doi.org/10.1145/1281192.1281210
10. Davari N, Veloso B, De Assis Costa G et al (2021) A survey on data-driven predictive maintenance for the railway industry. Sensors 21:5739. https://doi.org/10.3390/s21175739
11. Ester M, Kriegel HP, Sander J et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd international conference on knowledge discovery and data mining. AAAI Press, Portland, Oregon, KDD'96, pp 226–231
12. Feng X, Weng C, He X et al (2019) Online state-of-health estimation for li-ion battery using partial charging segment based on support vector machine. IEEE Transactions on Vehicular Technology 68(9). https://doi.org/10.1109/TVT.2019.2927120
13. Forestiero A, Pizzuti C, Spezzano G (2009) FlockStream: a bio-inspired algorithm for clustering evolving data streams. In: 2009 21st IEEE international conference on tools with artificial intelligence. pp 1–8, https://doi.org/10.1109/ICTAI.2009.60, iSSN: 2375-0197

14. Gomes HM, Read J, Bifet A et al (2019) Machine learning for streaming data: state of the art, challenges, and opportunities. ACM SIGKDD Explorations Newsl 21(2):6–22. https://doi.org/10.1145/3373464.3373470
15. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
16. Inturi V, Shreyas N, Chetti K et al (2021) Comprehensive fault diagnostics of wind turbine gearbox through adaptive condition monitoring scheme. Appl Acoust 174:107738
17. Järvelin K, Kekäläinen J (2002) Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20(4):422–446. https://doi.org/10.1145/582415.582418
18. Kranen P, Assent I, Baldauf C et al (2009) Self-adaptive anytime stream clustering. In: 2009 9th IEEE international conference on data mining. pp 249–258. https://doi.org/10.1109/ICDM.2009.47, ISSN: 2374–8486
19. Le Nguyen MH, Turgis F, Fayemi PE et al (2021) A complete streaming pipeline for real-time monitoring and predictive maintenance. In: Proceedings of the 31st European safety and reliability conference. pp 2119, https://doi.org/10.3850/978-981-18-2016-8_400-cd
20. Lebold M, Reichard K (2002) OSA-CBM architecture development with emphasis on XML implementations. In: OSA-CBM architecture development with emphasis on XML implementations
21. Li Y, Li H, Wang Z et al (2020) ESA-Stream: efficient self-adaptive online data stream clustering. IEEE Transactions on Knowledge and Data Engineering pp 1–1. https://doi.org/10.1109/TKDE.2020.2990196, conference Name: IEEE Transactions on Knowledge and Data Engineering
22. Lin J, Keogh E, Lonardi S et al (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. Association for Computing Machinery, New York, NY, USA, DMKD '03, pp 2–11, https://doi.org/10.1145/882082.882086
23. Liu Lx, Huang H, Guo Yf et al (2009) rDenStream, A clustering algorithm over an evolving data stream. In: 2009 international conference on information engineering and computer science. pp 1–4, 10.1109/ICIECS.2009.5363379, iSSN: 2156-7387
24. Mitici M, Hennink B, Pavel M et al (2023) Prognostics for Lithium-ion batteries for electric Vertical Take-off and Landing aircraft using data-driven machine learning. Energy and AI 12:100233
25. Polikar R (2012) Ensemble learning. In: Zhang C, Ma Y (eds) Ensemble machine learning: methods and applications. Springer US, Boston, MA, pp 1–34. https://doi.org/10.1007/978-1-4419-9326-7_1
26. Putina A, Rossi D (2021) Online anomaly detection leveraging stream-based clustering and real-time telemetry. IEEE Transactions on Network and Service Management 18(1):839–854. https://doi.org/10.1109/TNSM.2020.3037019, conference Name: IEEE Transactions on Network and Service Management
27. Ren J, Ma R (2009) Density-based data streams clustering over sliding windows. In: 2009 6th international conference on fuzzy systems and knowledge discovery. pp 248–252. https://doi.org/10.1109/FSKD.2009.553
28. Ribeiro RP, Pereira P, Gama J (2016) Sequential anomalies: a study in the railway industry. Mach Learn 105(1):127–153. https://doi.org/10.1007/s10994-016-5584-6
29. Ruiz C, Menasalvas E, Spiliopoulou M (2009) C-DenStream: using domain knowledge on a data stream. In: Gama J, Costa VS, Jorge AM et al (eds) Discovery science. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, pp 287–301, https://doi.org/10.1007/978-3-642-04747-3_23
30. Sahal R, Breslin JG, Ali MI (2020) Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case. J Manuf Syst 54:138–151
31. Settles B (2009) Active learning literature survey. Technical Report, University of Wisconsin-Madison Department of Computer Sciences
32. Shen J, Li S, Jia F et al (2020) A deep multi-label learning framework for the intelligent fault diagnosis of machines. IEEE Access 8:113557–113566. https://doi.org/10.1109/ACCESS.2020.3002826, Conference Name: IEEE Access
33. Su CJ, Huang SF (2018) Real-time big data analytics for hard disk drive predictive maintenance. Computers & Electrical Engineering 71:93–101
34. Tian H, Khoa NLD, Anaissi A et al (2019) Concept drift adaption for online anomaly detection in structural health monitoring. In: Proceedings of the 28th international conference on information and knowledge management, CIKM '19. pp 2813–2821, https://doi.org/10.1145/3357384.3357816
35. Torkamani S, Lohweg V (2017) Survey on time series motif discovery. WIREs Data Mining and Knowledge Discovery 7(2):e1199. https://doi.org/10.1002/widm.1199, https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1199
36. Turgis F, Audier P, Nemoz V et al (2022) Health state characterization using clustering algorithms for railway maintenance. In: World Congress on Railway Research 2022, Birmingham, United Kingdom
37. Zhao R, Yan R, Chen Z et al (2019) Deep learning and its applications to machine health monitoring. Mech Syst Signal Process 115:213–237
38. Zonta T, da Costa CA, da Rosa Righi R et al (2020) Predictive maintenance in the Industry 4.0: A systematic literature review. Computers & Industrial Engineering 150:106889. https://doi.org/10.1016/j.cie.2020.106889, http://www.sciencedirect.com/science/article/pii/S0360835220305787
39. Zubaroğlu A, Atalay V (2021) Data stream clustering: a review. Artif Intell Rev 54(2):1201–1236. https://doi.org/10.1007/s10462-020-09874-x

Authors and Affiliations

Minh-Huong Le-Nguyen[1,2] · Fabien Turgis[2] · Pierre-Emmanuel Fayemi[2] · Albert Bifet[1]

Fabien Turgis
fturgis@ikosconsulting.com

Pierre-Emmanuel Fayemi
pefayemi@ikosconsulting.com

Albert Bifet
albert.bifet@telecom-paris.fr

[1] INFRES, Telecom Paris, 19 Place Marguerite Perey, Palaiseau 91120, France

[2] Ikos Lab, IKOS Consulting, 155 Rue Anatole France, Levallois-Perret 92300, France