Software and Systems Modeling
https://doi.org/10.1007/s10270-021-00952-4

THEME SECTION PAPER

Event-driven temporal models for explanations - ETeMoX: explaining reinforcement learning

Juan Marcelo Parra-Ullauri[1] · Antonio García-Domínguez[1] · Nelly Bencomo[2] · Changgang Zheng[3] · Chen Zhen[4] · Juan Boubeta-Puig[5] · Guadalupe Ortiz[5] · Shufan Yang[6,7]

Received: 2 April 2021 / Revised: 27 September 2021 / Accepted: 4 November 2021

Abstract
Modern software systems are increasingly expected to show higher degrees of autonomy and self-management to cope with uncertain and diverse situations. As a consequence, autonomous systems can exhibit unexpected and surprising behaviours. This is exacerbated due to the ubiquity and complexity of Artificial Intelligence (AI)-based systems. This is the case of Reinforcement Learning (RL), where autonomous agents learn through trial-and-error how to find good solutions to a problem. Thus, the underlying decision-making criteria may become opaque to users that interact with the system and who may require explanations about the system's reasoning. Available work for eXplainable Reinforcement Learning (XRL) offers different trade-offs: e.g. for runtime explanations, the approaches are model-specific or can only analyse results after-the-fact. Different from these approaches, this paper aims to provide an *online model-agnostic* approach for XRL towards trustworthy and understandable AI. We present ETeMoX, an architecture based on temporal models to keep track of the decision-making processes of RL systems. In cases where the resources are limited (e.g. storage capacity or time to response), the architecture also integrates complex event processing, an event-driven approach, for detecting matches to event patterns that need to be stored, instead of keeping the entire history. The approach is applied to a mobile communications case study that uses RL for its decision-making. In order to test the generalisability of our approach, three variants of the underlying RL algorithms are used: Q-Learning, SARSA and DQN. The encouraging results show that using the proposed configurable architecture, RL developers are able to obtain explanations about the evolution of a metric, relationships between metrics, and were able to track situations of interest happening over time windows.

Keywords Temporal models · Complex event processing · Artificial intelligence · Explainable reinforcement learning · Event-driven monitoring

Communicated by L. Burgueño, J. Cabot, M. Wimmer & S. Zschaler.

✉ Juan Marcelo Parra-Ullauri
j.parra-ullauri@aston.ac.uk

Antonio García-Domínguez
a.garcia-dominguez@aston.ac.uk

Nelly Bencomo
nelly.bencomo@durham.ac.uk

Changgang Zheng
changgang.zheng@eng.ox.ac.uk

Chen Zhen
cz2016@mail.ustc.edu.cn

Juan Boubeta-Puig
juan.boubeta@uca.es

Guadalupe Ortiz
guadalupe.ortiz@uca.es

Shufan Yang
s.yang@napier.ac.uk

1 Engineering and Applied Science School, Aston University, Birmingham, United Kingdom

2 Department of Computer Science, Durham University, Durham, United Kingdom

3 Department of Engineering Science, University of Oxford, Oxford, United Kingdom

4 Computer Science and Technology, University of Science and Technology of China, Hefei, China

5 Department of Computer Science and Engineering, University of Cadiz, Cádiz, Spain

6 School of Computing, Edinburgh Napier University, Edinburgh, United Kingdom

7 Center of Medical and Industrial Ultrasonic, University of Glasgow, Glasgow, United Kingdom

Published online: 18 December 2021

1 Introduction

Artificial Intelligence (AI) aims to mimic cognitive functions for real-world problem solving, building systems that learn and think like people do [52]. The adoption of AI has become ubiquitous in software-based systems when needing to provide better levels of autonomy and self-management in modern software. A branch of AI is Machine Learning (ML), which has been successfully applied in vast domains such as transportation, recommendation systems or natural language processing among others [24,68,72]. Despite its broad applicability, the nature of ML is still considered as a "black-box" where system decisions can become opaque to stakeholders [12]. This is the case of Reinforcement Learning (RL), a ML technique where a system learns autonomously through a trial-and-error approach and whose decision-making criteria may lack transparency [54]. The insufficiency of validation techniques for the reasoning done by the system when using ML is a deterrent to broader adoption [61]: it is essential to improve the trustworthiness and understandability of AI-based systems [32]. *Explaining* the decision-making processes becomes increasingly important to enhance collaboration, and to increment confidence [35]. This is ratified by the General Data Protection Regulation (GDPR) law, which enshrines the right to explanation [10].

For the reasons pointed above, the development of more transparent AI, ML and RL models—we will call them *AI-models*[1] to differentiate them from the term model in Model-Driven Engineering (MDE)—has gained significant attention during the recent years [16]. Consequently, several terms have been coined, such as eXplainable AI (XAI) [61], explainable or interpretable ML [55] and eXplainable RL (XRL) [54]. The methods underpinning these terms aim to provide human-readable and interpretable explanations about the decisions taken by the algorithms involved [63]. The concerned tasks in the algorithms are usually performed by rule-based approximations of complex AI-models [54], aimed at better understanding what the AI-model has learned and/or explaining individual decisions [61].

In this paper, we focus on XRL for RL developers and RL-knowledgeable users (i.e. users who are experts in a given domain in which they apply RL, without necessarily being experts in RL). These users develop and operate software systems by applying RL AI-models and are, therefore, interested in understanding and diagnosing the results of the application of such AI-models in their domain. We argue that these RL users require tools to obtain explanations to explore the reasoning of RL-AI-models. These tools could be used to prove or disprove hypotheses posed on the system behaviour upon demand.

When studying the current state of the art of approaches for explanations [1,54], we noted that these approaches generally focused on the specific AI-model used [54]. In contrast to those approaches, this paper aims to provide an AI-model-agnostic approach for XRL.

As Bucchiarone et. al stated in [8], MDE techniques can help in the improvement of AI and machine learning. We argue that runtime models [5,6] can provide the abstraction, analysis and reasoning capabilities needed to support explanations when using AI-based systems. In our previous work [25,49], we proposed runtime models stored in Temporal Graph Databases (TGDB) which results in Temporal Models (TMs), for tracking the decision-making history of self-adaptive systems to support explanations about their behaviour. The approach allowed for explanations in both cases: interactive diagnosis (i.e. at runtime or during execution) and forensic analysis (i.e. after the system has finished its execution), based on the trajectory or history of the execution. We demonstrated how the approach offers substantial benefits, but at the cost of disk space and processing time. Nevertheless, these costs can be prohibitive when dealing with data-intensive systems, as it is the case of RL-based systems, where the volume and complexity of the data can grow considerably.

In this paper, we propose **ETeMoX** (**E**vent-driven **Te**mporal **Mo**dels for e**X**planations), a configurable architecture based on temporal models to keep track of the system's reasoning over time and to extract history-aware explanations on demand when using RL. In addition to TMs, the architecture integrates Complex Event Processing (CEP) [37], an event-driven technology for rapid detection of situations of interest. We use CEP to tackle the challenges associated with data-intensive systems. It serves as a *real-time filter* that selects relevant points in time that require to be stored in the TGDB as runtime models. The criteria for storing the system's history can be configured through event patterns on a CEP engine. For example, a certain data rate can be imposed, or the history may only keep points in time where certain conditions are met instead of the full history, saving memory resources.

The approach has been applied to a mobile communications case study using autonomous airborne base stations [70]. The system uses RL for positioning the simulated drones while maximising the covered end users. In order to test the AI-model agnosticism offered by **ETeMoX**, three variants of the underlying RL algorithm have been used: Q-Learning, State-Action-Reward-State-Action (SARSA) and Deep Q-Network (DQN). We have tested different filtering criteria and evaluated the results. Costs of storing and retrieving the system's history as well as the accuracy of the explanations provided have been analysed. The experiments

[1] AI-models are mathematical algorithms that are "trained" using data and human expert input to replicate a decision an expert would make when provided that same information [67].

were performed during the training of the AI-models, to help developers gain insights about the learning process while they work on validating and improving their systems. In summary, the main contributions of this work are as follows:

- ETeMoX, a generalisable framework based on TMs and CEP for the extraction of explanations from goal-oriented systems, as it is the case of RL systems.
- Evaluation of the results obtained in the experiments performed using different ETeMoX's configurations, based on the costs and accuracy.
- Demonstrating how event-driven temporal models can be used to provide information needed for post hoc local explanations within the XAI umbrella.

The ETeMoX framework, and the user manual are available at [47]. The rest of the paper is structured as follows. Sect. 2 presents the foundations that underlie this research. Section 3 describes ETeMoX and each component of our proposed architecture. The case study and the application of the architecture are presented in Sect. 4. Section 5 presents the RQs and the experiments. Then, the results of the experiments are discussed in Sect. 6. Afterwards, Sect. 7 compares this approach with related work. Finally, Sect. 8 presents the conclusions and future work.

2 Background

This section presents the core concepts and technologies used in our proposed architecture.

2.1 Artificial intelligence, machine learning and reinforcement learning

The ability to learn is one of the most fundamental attributes of intelligent behaviour [41]. AI is a broad scientific discipline with its roots in philosophy, mathematics and computer science that aims to understand and develop systems that display properties of intelligence [46]. ML is a sub-discipline of AI that focuses on building AI-models of human learning and understanding how machines can be empowered with the ability of learning [41]. In ML, agents [2] learn either from training data or from policies to create AI-models with minimal or no human intervention [38,46].

The focus of this paper is on reinforcement learning (RL), a particular type of ML method. Influenced by behavioural psychology [46], RL is an approach where agents learn actions based on their ability to maximise a defined reward in a trial-and-error fashion [54]. In RL, an agent is trained

[2] Agent: autonomous or semi-autonomous AI-driven system, in other words, the learner [67].

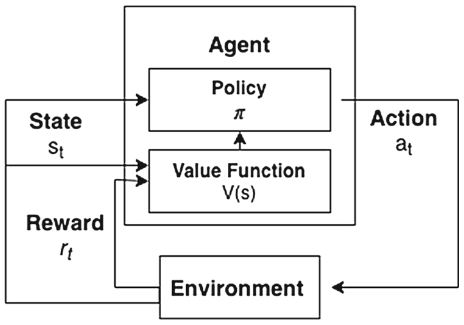


Fig. 1 Reinforcement learning

to select actions to interact with the environment that maximise the cumulative reward resulting from those interactions [38]. RL is usually introduced as a Markov Decision Process (MDP), as it satisfies the Markov property of sensation, action, and goal [67].

As shown in Fig. 1, RL agents learn in a continuous process, where interactions with the environment occur at discrete time steps (t). The agent initiates the learning process by performing a random action (a_t) that leads to a certain environmental state (s_t). The reward (r_t) corresponding to this state is assigned depending on how desirable this outcome is. After several iterations, the agent will learn a certain policy (π) (a function that maps states to actions), and will update the *value function* $V(s)$ or *action-value function* $Q(s, a)$ in order to maximise the cumulative reward, with the aim to select the optimal action in each situation in order to achieve long-term goals [38,67]. There are various algorithms for RL: Sect. 4 will discuss popular approaches (Q-Learning, SARSA, and DQN) and apply them to a case study.

2.2 Motivations for explanations in artificial intelligence

In the case of humans, explanations provide a key capability to shape the understanding that they develop when processing the environment, especially when their perceptions diverge from their expectations [16]. There are different arguments in favour of explanations in AI. Adadi et. al. stated four arguments in [1]:

- *Explain to justify* AI is involved in more and more areas of our everyday lives. People affected by AI-influenced decisions (e.g. when refused a loan) may demand a justification for the particular outcome. This transparency is needed to ensure fair and ethical decisions [69] are being made.
- *Explain to control* Explanations can often be used to keep agent actions inside an envelope of good behaviour. The explanations allow to discover the origin of a problem or

to clarify misunderstandings between the system and the user [2]. Indeed, explanations can contribute to prompt identification of errors in non-critical situations [1].

- *Explain to discover* Modern AI systems can process large amounts of data that otherwise would be difficult for humans to process. Asking for explanations is a helpful tool to extract insights about the knowledge acquired by this processing [1].
- *Explain to improve* In order to improve an AI system, it is key to discover its flaws. An AI-model that can be explained and understood can be easier to enhance and use to the best advantage [61].

In this paper, the focus is on *explanations to control* and *explanations to discover* for RL developers and RL knowledgeable users. These two groups of users are familiar with developing and/or using RL AI-models and are, hence, interested in understanding, diagnosing, as well as refining such AI-models in a given application context [29].

2.3 Common approaches in XAI/XRL

There are different techniques and approaches that are proposed to confer explainability. In RL, these can be classified broadly across two dimensions [29,54,54] (Fig. 2):

(1) *Depending on when the information is extracted*, it can be done beforehand (a.k.a. intrinsic) or in a post hoc fashion. An explanation could be extracted and/or generated intrinsically or post hoc. The most straightforward way to get an interpretable AI-model is to make it intrinsically explainable, thus self-explainable at the time of training [1,54]. One example is decision-trees: they have a defined structure and can provide convincing capabilities to gain the trust of domain experts [33]. These types of explanations are AI-model-specific by definition [1]. Post hoc explainability aims to mimic the original AI-model to provide the needed explanations without altering or even knowing the inner works of the original AI-model [36]. Rule extraction is an example of this type. By analysing the input and output of an artificial neural network, it provides a description of the knowledge learned by the network during its training by extracting rules that approximate the decision-making processes [55]. This type of explanations is generally AI-model-agnostic and is generated and/or received after training [1,54].

(2) *Depending on the scope of the explanations*, it can be global or local [54]. Local explanations focus on data and provide individual explanations, helping provide trust on AI-model outcomes. Local explanations focus on why did the AI-model make a certain decision for one or a group of instances [54], whereas global explanations focus on the AI-model and provide an understanding of the overall decision process. A global explanation aims to provide a general understanding of how the AI-model works [1].

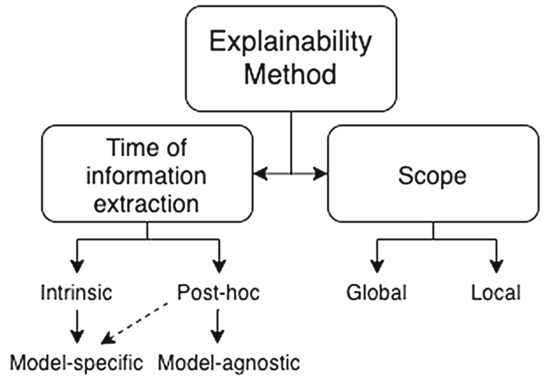


Fig. 2 XAI methods taxonomy [54]

In Sect. 3, we explain where our architecture sits along these two dimensions.

2.4 Historical data management

Identifying historical patterns in the data produced by a system has been a topic of interest for a long time. A 2012 survey on time-series mining by Esling et al. [21] outlines more than two decades of research work on this topic. Typical tasks include finding timepoints of interest, clustering similar regions, classifying timepoints, finding anomalies or predicting future timepoints.

In regard to industrial applications, the need to organise the large volumes of data generated by the Web and the Internet of Things has motivated the development of better time-series analysis capabilities in database technologies. For instance, the Elasticsearch search engine can index large document collections with numerical measurements over time, and then apply machine learning approaches to find anomalies [20].

Still, these time-series have the limitation that they simply track the evolution of a metric: they cannot track, for instance, the evolution of the relationships within a system. They cannot directly represent the relationships between multiple evolving metrics, either. Graph databases such as Neo4j [57] are designed specifically to represent complex networks of relationships: their data is structured into nodes connected by edges. Nodes and edges have a label (e.g. “sensor”) and a set of key/value pairs (e.g. “lastReading”). Graph databases have been successfully used for representing transport networks, social networks and other similarly interconnected systems. However, they do not explicitly model the time dimension.

Different extensions to graph databases exist to introduce the time axis: these TGDBs record how nodes and edges appear, disappear and change their key/value pairs over time. Some of these proposals include Greycat [28] from Hartmann et al. and Chronograph [27] from Haeusler et al. In particular, Greycat is an open-source solution which reuses several existing database engines (e.g. the LevelDB key/value store) to implement a TG data model. Nodes and edges in Greycat have a lifespan: they are created at a certain timepoint, they may change in state over the various timepoints, and they may be "ended" at another timepoint. Greycat considers edges to be part of the state of their source and target nodes. It also uses a copy-on-write mechanism to store only the parts of a graph that changed at a certain timepoint, to therefore save disk space. In the present work, we use TGDB to track the evolution of the decision-making processes.

2.5 Event-driven monitoring

Event-driven monitoring allows us to detect the occurrences of predefined events on one or multiple incoming data streams, in order to be notified of their occurrence and/or run some palliative processes. There are several proposals for event-driven monitoring in the literature; for instance, Konno et al. work [31] uses a rule inference method that integrates dynamic case-based reasoning and root cause analysis. It allows for autonomous recovery and failure prevention to guarantee long-term QoS of cloud systems. Other event-driven monitoring approaches integrate Wireless Sensor Networks (WSNs) with sentinel nodes [22,53] to detect heavy road vehicles as well as raising alarms in monitoring nodes. CEP has been widely used for real-time event-driven monitoring [3,48]. CEP can capture, analyse and correlate large amounts of data in real time across many application domains [37]. The main objective is to detect situations of interest in a specific domain or scenario [7]. In order to detect such situations, it is needed to previously define a set of event patterns that specify the conditions that incoming events to the system must fulfil in order to match the situation of interest. An incoming event can be *simple* (something that happens in the system at a point in time) or *complex* (patterns of two or more events that happen over a period of time). Any detected complex events can be fed back to the CEP system for further matching which creates a hierarchy of complex events types [48]. The defined patterns must be deployed to a CEP engine, i.e. the software that allows the incoming data streams to be analysed in real time according to the defined patterns [7]. Each CEP engine provides its own Event Processing Language (EPL) for defining the patterns to be deployed.

In this paper, we integrate CEP into the proposed architecture to allow us to detect events that will conform our TMs. Among the existing CEP engines, we opted for Esper[3], a mature, scalable and high-performance CEP engine. The Esper EPL is a language similar to SQL but extended with temporal, causal and pattern operators, as well as data windows. Upon matching a pattern, a complex event summarising the detected situation will be created and then notified to the interested event consumers, such as dashboards, databases, services and actuators. Indeed, CEP has been widely used for real-time event monitoring in various software architectures and application domains [7,15,58].

[3] https://www.espertech.com/esper/.

3 Proposal: ETeMoX framework

This section presents the architecture of *ETeMoX*, which integrates CEP and TMs to support the generation of explanations for RL-based systems. Based on the categorisation from Sect. 2.3, we aim to build an architecture for AI-model-agnostic post hoc explainability, using the benefits of event-driven monitoring and model-driven engineering. The architecture targets AI-models that are not interpretable by design. It focuses on local explanations to promote an understanding on why the AI-model made specific decisions for a group of instances. Understanding what the system did requires: (1) the system to track its own decision history, and (2) to explain those decisions to the users coherently. Both requirements are presented in this work.

As shown in Fig. 3, there are four components in the framework: *Translator*, *Filter*, *Temporal Model* and *Explainer*. These will be described in detail below.

3.1 Translator component

Our implementation decouples the decision-making processes in the system from the generation of the explanations. The translator component receives data streams with execution traces. The traces (Logs) contain information related to the observations made by the agent about its decisions, actions, states, rewards, and environment. The monitored system collects and exposes the data streams to the translator component through a message broker ("A" in Fig. 3). An example of a broker is the open source Eclipse Mosquitto MQTT message broker [34]. This broker uses a publish-subscribe connectivity protocol, where messages are published according to a set of topics and users subscribe to the topics of their interest and it is used for lightweight messaging. The log can follow structured (JSON/XML) or unstructured (plain text) formats: we have selected JSON for this implementation. This JSON log containing unprocessed data is converted into the data format required by the CEP engine, and then inserted into the Filter component for processing ("B" in Fig. 3).

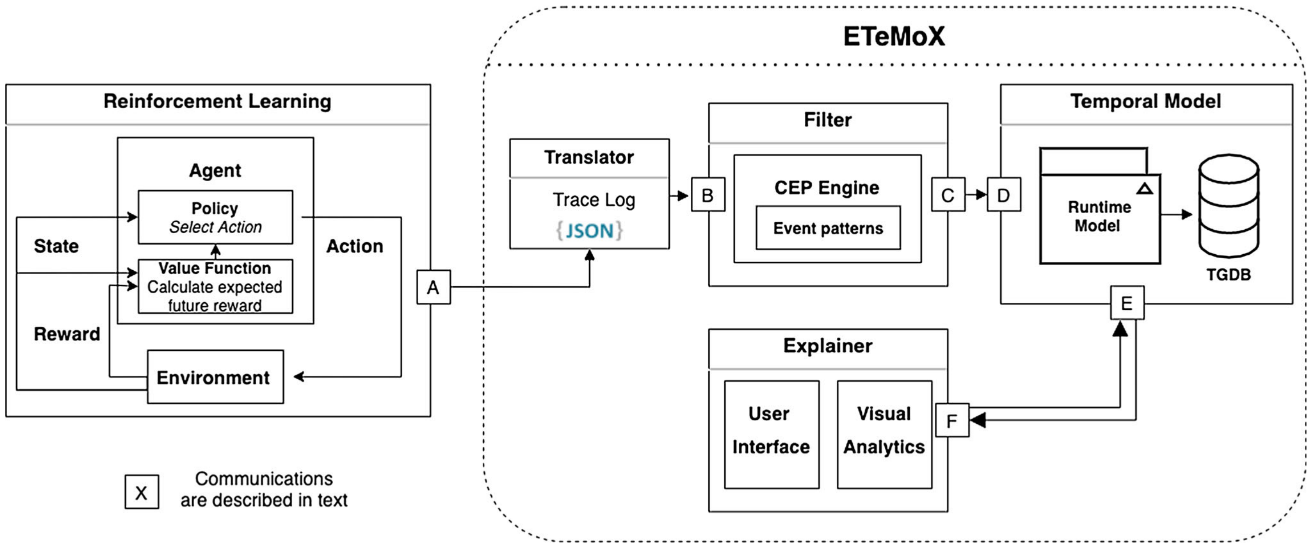


Fig. 3 Event-Driven temporal models for explanations (ETeMoX) architecture

3.2 Filter component

This component performs the transformation, processing, analysis and routing of data from the Translator component to the Temporal Model component. The main element in this component is a CEP engine. As mentioned in Sect. 2.5, we selected Esper as the CEP engine. Esper processes and correlates the simple events coming from the Translator component, aiming to detect in real-time situations of interest that will match the filtering criteria. As previously explained, these situations of interest are described through event patterns. Developers define the focus of interest, i.e. the subset of the data that will be recorded in the TGDB. Event patterns are implemented in Esper EPL and deployed to the Esper engine. When the filtering conditions are met (i.e. patterns are detected), the engine automatically generates complex events that collect the required information, and sends them to the Temporal Model component. The communication from the Filter component to the Temporal Model component ("C" to "D" in Fig. 3) is performed using a message broker similar to the one employed by the Translator component.

3.3 Temporal model component

The incoming complex events containing the log information about the state of the system are reshaped into the trace metamodel (Fig. 4 from [49]) based on the Eclipse Modeling Framework[4] (EMF) for linking the system goals and decisions to its observations and reasoning. This metamodel is divided into two parts: (1) a generic part, for goal-oriented autonomous systems, and (2) a more specific part, for systems that take into account Non-Functional Requirements (*NFR*) and their satisfaction (*NFRSatisfaction*) to support the system's decision-making. The top half of Fig. 4 represents the general part. In the trace metamodel (Fig. 4), the *Log* records the *Decision*s to be made, the available *Action*s and their beliefs (i.e. probabilities for satisfying a functional requirement) and the *Observation*s of the environment. Each *Decision* is based on an *Observation* of the environment, which produces a set of *Measurement*s of the *Metric*s. Different types of measurements are allowed (e.g. *DoubleMeasurement* and *StringMeasurement*). The runtime model will then be used to update the TGDB, creating a new snapshot at the current point in time: all relevant versions are kept. We use a model indexer to automatically compare the runtime model as an object graph against the current version of the temporal graph. It creates a new version which only updates the temporal graph where needed, for efficient storage. Specifically, ETeMoX uses Eclipse Hawk[5], which operates on Greycat temporal graphs. By using TGDBs, it is possible to track the evolution of certain metrics at each node, as well as the changes in the relationships of the various entities in the system, or their appearance and disappearance.

3.4 Explainer component

This component is where the explanations are constructed and presented. The explainer component can run a query

[4] https://www.eclipse.org/modeling/emf/.

[5] https://www.eclipse.org/hawk/.

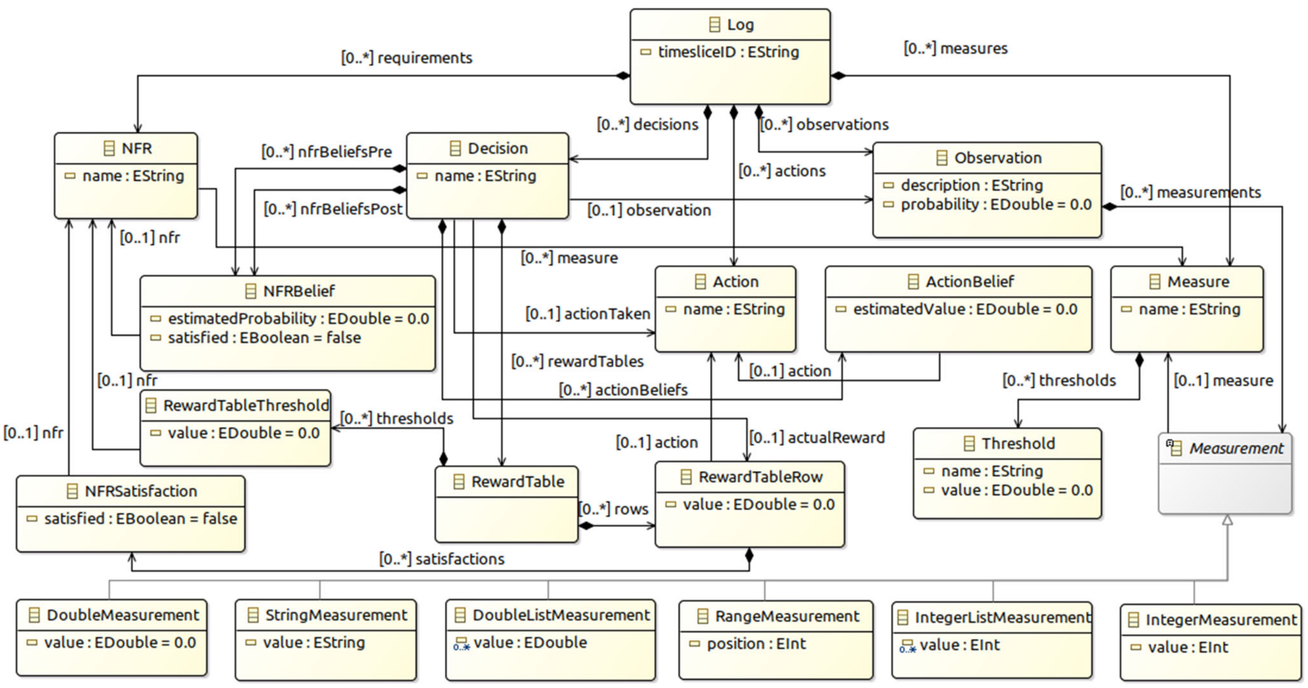


Fig. 4 Execution trace metamodel for a decision-based self-adaptive system [49]

on the TGDB using our time-aware query language, an extension of the Epsilon Object Language (EOL) to define temporal patterns that traverse the history of a model. The result of this query contains the information that will be used to construct the explanations. These explanations could be presented in textual or graphical ways, e.g. plots of various kinds, yes/no answers, or specific examples of matches of a certain temporal pattern.

In relation to the explanation phases defined in [2], this work tackles the first two: i) the *explanation generation* is the construction of the causally connected TGDB (performed on the previous component) ii) the *explanation communication* is the extraction of the information using the temporal query language (what information will be provided) and the presentation of explanations either textually or graphically (how will it be presented).

In order for an explanation to satisfy its recipient, it needs to be expressed in a way that is easy to understand for that recipient. Therefore, a rigid system for which developers or domain expert have defined explanations with no awareness of the needs and expectations of the recipients may be not convenient for users with different backgrounds. The Explainer component in ETeMoX allows users to specify their own custom queries over the historic behaviour of the system, helping the users to complete their mental model of how the system works, or test hypotheses about its behaviour. This is done by forwarding the queries to the query engine in the Eclipse Hawk model indexer through the Hawk API (the "E–F" communication in Fig. 3).

3.5 ETeMoX step-by-step guideline

As mentioned above, the user manual that explains how to apply ETeMoX to an RL system can be found in the GitLab project at [47]. In summary, to run the implementation, a user requires: (1) an RL system that exposes its decision-making traces, (2) a parser to translate these traces into the metamodel in Fig. 4, (3) a set of event patterns that define the filtering criteria (and their deployment to the Esper CEP engine), (4) a Hawk instance indexing the translated traces into a temporal model, and (5) a set of temporal queries that extract the history-aware explanations from the temporal model. The detailed step-by-step guidelines are as follows:

1. The proposed post hoc approach is designed to be as least intrusive as possible for the RL agent to be explained. The first step is to collect observations from the system's decision-making. The available information about the agent's states, rewards, actions, and environment is exposed to the system through logs, which may be structured or not.
2. Once this log is constructed, the next step is to feed it to our architecture ("A" in Fig. 3). We use MQTT as the core communication protocol, with MQTT clients in the different components talking to a central MQTT broker. The trace log is published to a MQTT topic to which the *translator* component is subscribed.
3. In order to handle the incoming data (i.e. trace log), ETeMoX requires the *translator* component to parse the log.

In the current implementation, this parser has been manually defined at design time, and is specific for each case study. However, different techniques for log and stream pre-processing are being studied for automatically creating these translators, such as the one proposed by Corral-Plaza et al. in [15].

4. When the parser that processes the log is ready, the next step is the creation of the event patterns needed by the *filter* component. These EPL patterns will contain the criteria to curate the data, based on events of interest. Some predefined filtering criteria can be reused across projects (e.g. sampling at a certain rate). Also, problem-specific event patterns of interest can be added as needed. After, the filtered data is sent to the *Temporal Model* component over MQTT.
5. Next, the filtered information is stored in a causally connected and efficient way in an Eclipse Hawk instance. This instance needs to be specified in order to use the execution trace metamodel of Fig. 4 as structure and a Greycat temporal graph database as backend.
6. Once the information is structured as a Temporal Model, it is possible to extract information for explanations using EOL queries from the *Explainer* component. Depending on the requirements, some predefined temporal queries can be reused, or new domain-specific queries can be formulated. Information for explanations can be extracted after-the-fact as in [25], or at runtime as in [49].
7. The final step is to construct an explanation from the queries. The specific way this is done depends on the requirements and the targeted audience. For instance, textual explanations (logs or natural language) or visual ones (graphs, plots or heatmaps) could be used.

An implementation of the proposed approach is presented in Sect. 4, based on a case study from the domain of RL-based self-adaptive systems.

4 Case study: autonomous airborne base stations

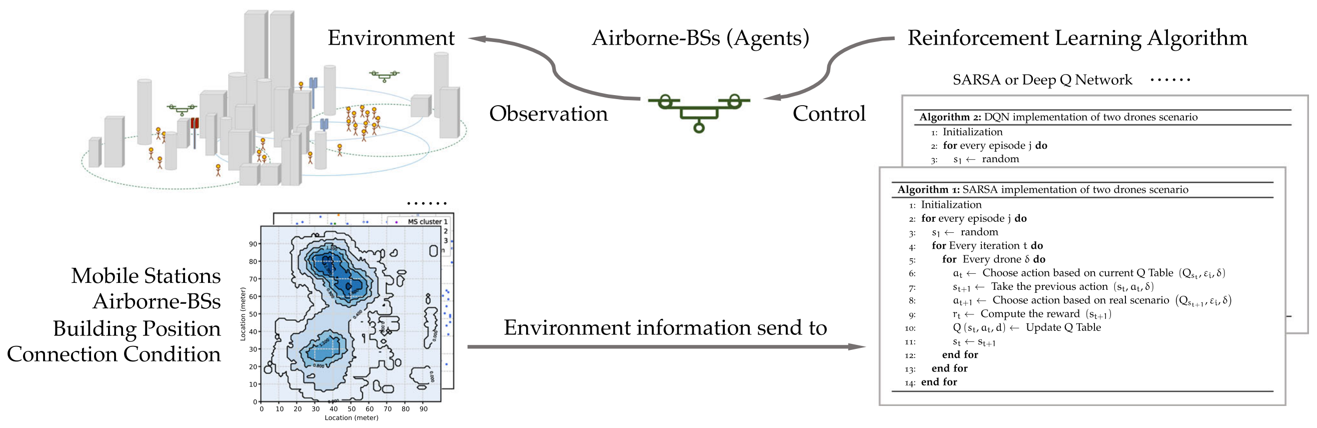
In order to demonstrate the feasibility of the proposed architecture, this section presents its implementation for a case study from the domain of mobile communications. In this case study, Airborne Base Stations (ABSes) use RL AI-models to decide where to move autonomously in order to provide connectivity to as many users as possible. The developers of the system are interested in studying the reasons why the AI-model acted as it did, both regarding single decisions and regarding its overall performance.

The rest of this section describes the system and the different RL AI-models used in this case study. This is followed by the explanation requirements for this case study, which motivates the chosen experimental approach for its evaluation.

4.1 Description of the case study

Mobile connectivity requires that an adequate network of base stations has been set in place. When these networks cannot meet unexpected spikes with respect to user demand [18,40] (e.g. due to a large concentration of users in one place, or due to failures in nearby communication towers), a swarm of ABSes can act as a backup. The goal of the system under study (the ABS Self-Adaptive System [76]) is to precisely control the location of the ABSes in relation to the locations of users and the other stations, trying to serve as many users as possible while ensuring high-signal strength and minimising interference among ABSes [77].

Figure 5 outlines the components of the ABS SAS case study. The environment is simulated according to a Mobile Station Distribution Model, and a 5G Communication System Model [76]. The mobile station distribution model simulates a random population of users that require connec-



```
Algorithm 2: DQN implementation of two drones scenario
Initialization
for every episode j do
    $s_1 \leftarrow$ random
```

```
Algorithm 1: SARSA implementation of two drones scenario
Initialization
for every episode j do
    $s_1 \leftarrow$ random
    for Every iteration t do
        for Every drone $\delta$ do
            $a_t \leftarrow$ Choose action based on current Q Table $(Q_{s_t}, \varepsilon_i, \delta)$
            $s_{t+1} \leftarrow$ Take the previous action $(s_t, a_t, \delta)$
            $a_{t+1} \leftarrow$ Choose action based on real scenario $(Q_{s_{t+1}}, \varepsilon_i, \delta)$
            $r_t \leftarrow$ Compute the reward $(s_{t+1})$
            $Q(s_t, a_t, d) \leftarrow$ Update Q Table
            $s_t \leftarrow s_{t+1}$
        end for
    end for
end for
```

Fig. 5 Overview of the ABS SAS case study

tivity on their end-user devices (e.g. mobile phones). This simulation randomly places users following a bivariate distribution over their latitude and longitude [56], involving the probability by which they appear, the number of mobile stations they carry with them, and their location.

The 5G Communications System Model performs the necessary calculations to estimate the Signal-to-Interference-plus-Noise Ratio (SINR) and the Reference Signal Received Power (RSRP) [76]. The SINR and RSRP values measure the signal quality of the communications between the ABSes and the mobile stations. SINR and RSRP thresholds are used to determine whether a station can be considered to be "connected" or not.

The appendix at the end of this paper describes in detail the simulation environment using SARSA, one of the RL AI-models [70]. The other two RL AI-models (Q-learning and DQN) are used in the same simulation environment with learning methods, as defined below.

4.1.1 Q-Learning

Q-Learning is an RL algorithm where an agent uses an action-value function $Q(s, a)$ to evaluate the expectation of the maximum future cumulative reward. This reward r_t is obtained from different executions of an action a_t in a given state s_t [66], which provides agents with the capability of learning to act with the aim of maximising the global reward [74].

$$\text{New } Q(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma \cdot \max Q'\left(s', a'\right) - Q(s,a)\right] \quad (1)$$

Traditional Q-Learning uses a simple lookup table for calculating the maximum expected future rewards for an action at each state. It is often referred to as the Q-table, as it is a way of representing the Q-values (or Action-Values) in the Value function V_s [51]. Equation 1 is used to update the Q-table, where the α is the learning rate to determine how much of the sum of immediate rewards will be used. γ is the discount factor to determine the importance of future rewards and $R(s, a)$ is the reward of the action at state s_t. $Q'(s', a')$ is the new Q value in next time step; s' is next state of environment; a' is the next action that ABSes is planning to take.

4.1.2 State-action-reward-state-action (SARSA)

SARSA is an RL algorithm very similar to Q-Learning [66, 75]. The main difference between SARSA and Q-Learning algorithms is the policy (π) type. SARSA, as a typical on-policy algorithm, have agents directly inferring a policy. On the contrary, Q-learning uses Q-values to quantify the value of every state-action pair [67].

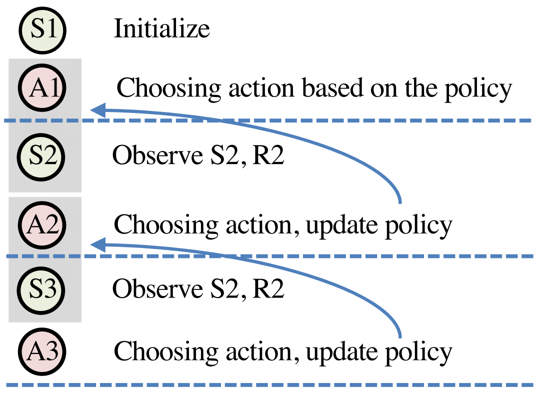

S1
Initialize
A1
Choosing action based on the policy
S2
Observe S2, R2
A2
Choosing action, update policy
S3
Observe S2, R2
A3
Choosing action, update policy

Fig. 6 Q-learning flow chart

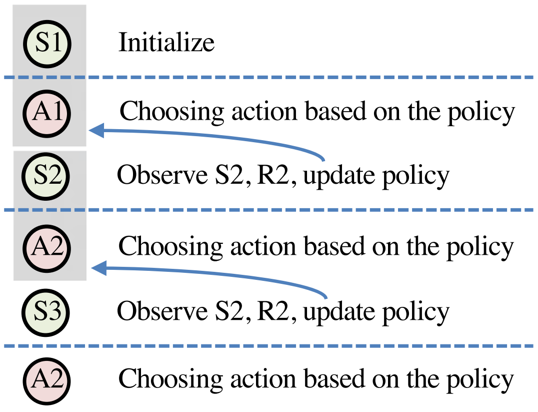

S1
Initialize
A1
Choosing action based on the policy
S2
Observe S2, R2, update policy
A2
Choosing action based on the policy
S3
Observe S2, R2, update policy
A2
Choosing action based on the policy

Fig. 7 SARSA flow chart

$$\text{New } Q(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma \cdot Q'\left(s', a'\right) - Q(s,a)\right] \quad (2)$$

In Eq. 2, α is the learning rate, γ is the discount factor and $R(s, a)$ is the reward of the action at state s. Figures 6 and 7 show the flow chart related to the update process of the action, the state and the value function at each iteration. The circles refer to the updated pairs of states and actions. S1, S2 and S3 indicate the state of environment; while A1, A2 and A3 indicate each stage of actions. Blue dash line indicates the pair $Q(s, a)$ updates in various stages.

The most important difference between Q-learning and SARSA is how $Q(s, a)$ is updated after each action. Although the update of $Q(s, a)$ in SARSA is quite similar to Q-learning, both algorithms have different ways of choosing actions. SARSA uses the behaviour policy (meaning, the policy used by the agent to generate experience in the environment randomly) to select an additional action a_{t+1}, and then uses $Q(s_{t+1}, a_{t+1})$ (discounted by γ) as the expected future return in the computation of the update action and state value [67]. Q-learning does not use the behaviour policy to select an additional action a_{t+1}. Instead, it estimates the expected future returns in the update rule as maximum

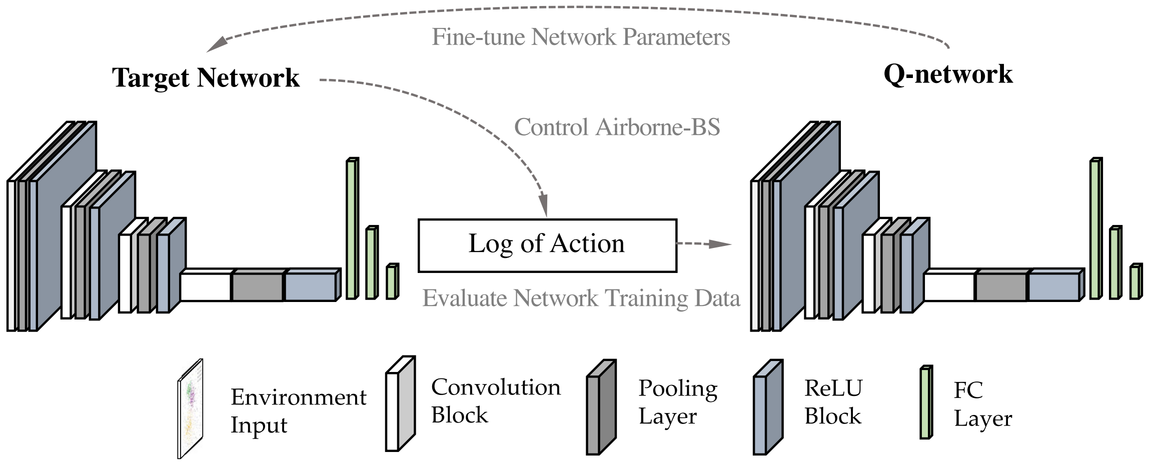


Fig. 8 Deep Q-network

action and state value. It is noted that Q-learning uses different policies for choosing the next action A2 and updating $Q(s, a)$. In other words, it tries to evaluate the policy while following the old policy; therefore it is seen as an off-policy algorithm. In contrast, SARSA uses the same policy all the time; hence it is seen as an on-policy algorithm.

4.1.3 Deep Q-network (DQN)

Q-learning and SARSA both use a lookup table (Q-table) to learn the best action to take. With large state or action spaces, the size of this lookup table can make learning intractable. In contrast, Deep Q Networks, avoid using a lookup table by instead predicting the Q-value of the current or potential states and actions using artificial neural networks [42]. As shown in Fig. 8, two convolutional neural networks (Q-Networks) constantly update their parameters to learn the optimal action to take: the target network, and the evaluate network.

A target network takes the current state and estimates the Q value of each action, and then the generated states, actions and rewards of each simulation step are saved to the memory pool (the “Log of Action” elements shown in Fig. 8). This target network is a copy of the action-value function (or Q-function) that is held constant to serve as a stable target for learning for some fixed number of time steps. The Q-network periodically updates the actions and action-state value $Q(s, a)$. The parameters of the target network are updated regularly by copying the weights from the Q-network, as shown in Fig. 8.

4.2 Explanation requirements for developers

Currently, generality is the biggest challenge for RL. Although many RL methods can be seen as performing well, it is difficult to apply them for generalisation purposes due to unforeseen situations [65]. Further, the traditional perception of RL methods is often viewed as black boxes. Without the proper tools, it is challenging to understand the behaviour of complex RL methods to solve general issues, especially when combining multiple neural networks for evaluating value functions during learning stages.

One way to improve the generalisation capabilities of RL is to dissect the reasons for failure during the learning stage. Often, the failure comes from the fact that RL AI-models have limited memory storage to store previous states, actions and $Q(s, a)$ values, which can be used later to estimate the value function. Sometimes, an action picked from the policy could lead to learning failure. Unfortunately, this can only be observed in retrospective after some time. Since developers can only investigate the overall reward of all agents after a certain number of time steps, it is extremely difficult to find out the situation when the RL AI-model has picked up a feature that may have been incorrectly used in the estimation of the value function.

The current trend of using deep neural networks presents new challenges for finding interpretable features that can be visualised, the first one being how to evaluate the unknown value function. In the ABS SAS case study, the spatial position of users and the signal interference from ABSes keep changing. Presenting the evolution of this change can help developers to understand if the ABS SAS is progressing towards its ultimate goal. Another important aspect for developers is to analyse the learning process and how the initial conditions affect it. Furthermore, in a multi-agent system, explaining collaborative aspects can help to understand whether the ABSes learn to coordinate to achieve the global goal or not. This can be difficult without the support of a third party like ETeMoX, considering that each RL AI-model has only access to its own local observation and is only responsible for choosing actions from its own action-state values.

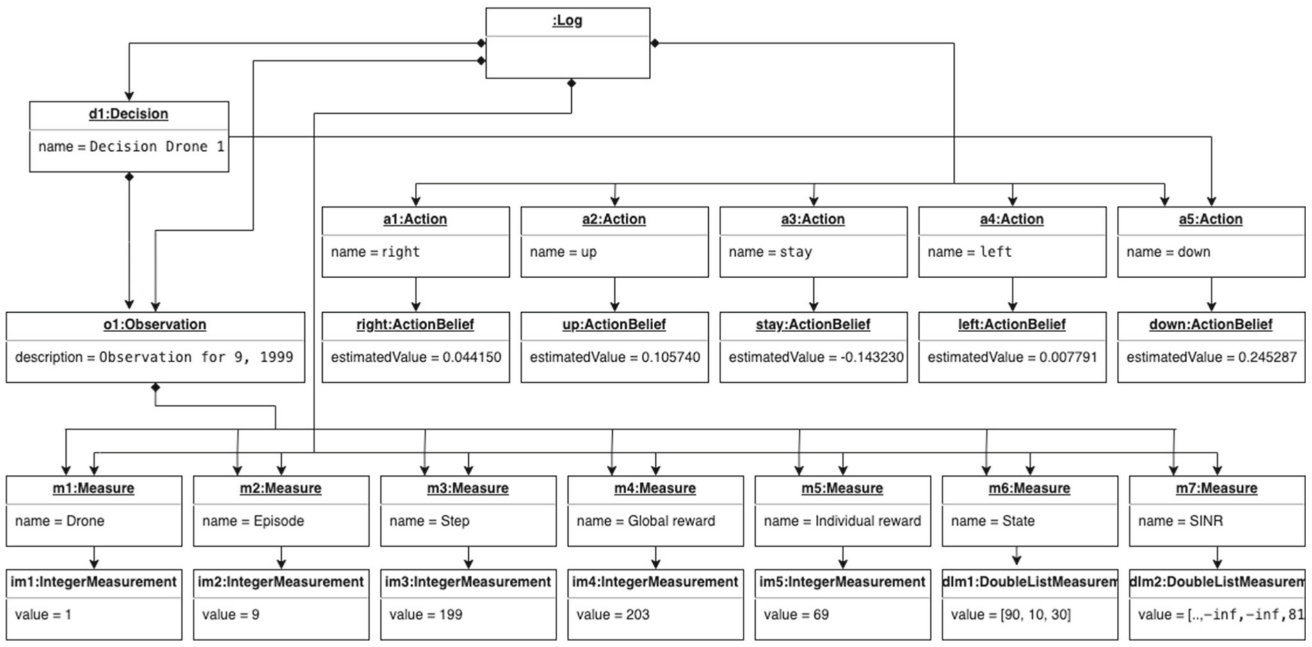


Fig. 9 Runtime model object diagram

4.3 Integration with ETeMoX

In order to evaluate whether the architecture proposed in Sect. 3 meets the requirements described in Sect. 4.2, the ABS SAS case study was integrated with our current implementation of ETeMoX. First, the three RL variants used in the ABS SAS case study (i.e. Q-learning, SARSA, and DQN) were extended to send their decisions and observations to a queue in a MQTT message broker in JSON format at each simulation step. The rest of the components of the ETeMoX architecture were implemented as follows:

- The Translator component receives and parses the messages, and sends them to the CEP engine in the Filter component as a *simple event*.
- The Filter component offers a number of *event patterns*, which act as the filtering criteria. Further details are provided in Sect. 5. An event match will trigger the creation of a *complex event*. This complex event is sent to the Temporal Model component through a different MQTT queue, using JSON format.
- The Temporal Model component receives complex events and records their information as a new version of the runtime model in the temporal graph database. Specifically, the Eclipse Hawk model indexer was extended with the capability to subscribe to an MQTT queue and reshape the information into a model conforming to the metamodel in Fig. 4. An object diagram with an instance of the runtime model at a certain step in the simulation is shown in Fig. 9. The *Log* contains *Decisions* and *Observations* for `ABS 1` at `Episode 9` and `Step 199`. The possible *Actions* are linked to their *ActionBelief* s that represent the estimated values (Q-values), which maximise the cumulative *Measure*: `Global reward` at the given *Measure*: `State`.
- Having recorded the history of the system so far, at any time the TGBD can answer queries from the explainer component. For the current implementation, we used the Graphical User Interface (GUI) of Hawk to extract the information needed to build the required explanations on demand, by using its dedicated time-aware query language [25].

5 Experiments

In this section, we will describe experiments performed with different filtering criteria, as well several types of explanations for the ABS SAS case study. The requirements from Sect. 4.2 are taken into account for explaining the ABSes system using ETeMoX. Through this study and considering the main objectives from Sect. 1, we aim to answer the following Research Questions (RQs):

- RQ1: How can TMs and CEP enable AI-model-agnostic XRL?
- RQ2: What types of explanations can be obtained using the proposed architecture integrating TM and CEP?

- RQ3: What are the costs of storing and retrieving explanations depending on the different configurations of the proposed architecture?
- RQ4: How accurate are the results derived from each configuration approach?

5.1 Experiment 1: evolution of a metric

The ABSes in the case study need to collaborate to maximise the number of connected users. As the RL AI-models analyse and update their decision-making criteria based on the rewards received at each time step, it is key for the explanatory system to keep track of (i.e. store) these rewards. Presenting the evolution of this metric to developers can help to understand if the ABSes system is progressing towards its main goal of maximising rewards.

ETeMoX is capable of tracking both the individual and global reward on every single time step in the system's training history, and present an average reward by episode. Without any filtering, the results would match exactly what the system experienced, but at high storage and processing costs. In order to answer RQ3 (costs involved) and RQ4 (accuracy obtained), we have defined three Esper EPL event patterns that apply various sampling rates, updating the TGDB every 10 steps of the simulation, every 100 steps, and every 500 steps. Listing 1 shows the event pattern for indexing the runtime model into the TGDB every 10 steps. The log about the state and observations of the system when these criteria are met is recorded.

Listing 1 EPL pattern to sample the data every 10 steps

```
@public @buseventtype @Name("Sampler")
insert into Sampler
select drone as complexEventInfo
from pattern [every drone = DronesLog(drone.step%10=0)]
```

5.2 Experiment 2: exploration versus exploitation

A common problem in RL is finding a balance between *exploration* and *exploitation*. Exploration means trying to discover new features of the environment by selecting a sub-optimal action. On the other hand, exploitation is when the agent chooses the best action according to what it already knows [14].

In order to find when a decision was performed using exploration or using exploitation, it is required to track the actual action taken and the Q-values (i.e. the *ActionBelief* s) for each possible action at given state. On one hand, when the action performed has the maximum Q-value then it could be said that the decision was taken by exploitation. On the other hand, if the action taken does not have the maximum Q-value, the action was taken by exploration. Considering the object diagram of Fig. 9, where the *Action* selected (represented by the reference from *d1* to *a5*) was *down*, it can be seen that it is the one with the maximum estimated value: thus, it can be concluded that the decision was performed by exploitation.

In order to evaluate the effect that a domain-specific filtering pattern could have on costs (RQ3) and accuracy (RQ4), we decided to create an Esper EPL pattern to only capture in the TGDB the moments when a decision was performed using exploration. Listing 2 shows the Esper EPL pattern for finding this situation. At every point in time the Q-value of the action selected (`drone.qtable.action`) is compared to the `maxValue()`, the action with the maximum Q-value. If these values don't match a decision was taken by exploration and the log about the state and observations of the system at that point in time is recorded. For this experiment, two TGDBs were created in parallel. One using the sampled data, and another that contains the full history of the system. In this last one, we ran the temporal-query of Listing 3 for validation. The temporal query written in EOL follows the same logic of the Esper EPL pattern, but traverses the full history. It looks for the Q-value of the action selected in each decision (`actionTakenValue`) and compares it with the maximum Q-value (`maxAB`). It returns a sequence of situations where the criteria are met. Finally, it reports a count of these situations.

Listing 2 EPL pattern to select when the system performs an action based on *exploration*

```
@public @buseventtype @Name("Exploration")
expression selectedActionValue{
        droneLog => case drone.qtable.action
          when "east" then drone.qtable.position.east
          when "west" then drone.qtable.position.west
          when "south" then drone.qtable.position.south
          when "north" then drone.qtable.position.north
          when "stay" then drone.qtable.position.stay
        end
}
expression maxValue{
        droneLog => max(drone.qtable.position.east,
                        drone.qtable.position.west,
                        drone.qtable.position.south,
                        drone.qtable.position.north,
                        drone.qtable.position.stay)
}

insert into Exploration
select drone as Log
from pattern [every drone = DronesLog] as droneLog
where maxValue(droneLog) != selectedActionValue(droneLog)
                        and maxValue(droneLog) != 0
```

Listing 3 EOL query to select when the system performed an action based on *exploration*

```
var results : Sequence;

for (decision in Decision.latest.all) {
  var totalCount = decision.versions.size;
  var exploration = decision.versions
                    .select(vlv.isExploration());
  var totalExploration = exploration.size;

  results.add(Map {
    'total' = totalCount,
    'exploration' = totalExploration
  });
}
return results;

operation Decision isExploration() : Boolean {
  var maxAB = self.actionBeliefs.estimatedValue.max();
  var actionTakenName=self.actionTaken.name;
  var actions = self.actionBeliefs.action;
  var actionTakenValue = actions
                .selectOne(ala.name = actionTakenName)
                .revRefNav_action.estimatedValue.first;
  return actionTakenValue <> maxAB and maxAB <> 0.0;
}
```

5.3 Experiment 3: collaborations within the multi-agent system

A challenge in mobile wireless communication is the hand-off or handover process. This is the process of providing continuous service by transferring a data session from one cell to another [50]. In this collaborative system, ABSes are assumed to have ideal communication among themselves and cannot occupy the same position (state) at the same time. Therefore, considering these conditions, handovers should be kept at a minimum, demonstrating effective communication and stability within the system.

In the present implementation, a handover could be considered when a user is connected to one ABS and then transferred to a different ABS in a short period of time. In order to find those situations in the different RL-algorithms, we ran a query on TGDBs containing the whole history of each RL-approach. Algorithm 1 describes the logic followed in the query, which was implemented in the temporal-query language supported by Hawk. A user (U) is connected to an ABS (D) when its received SINR ($\text{SINR}_{u,d}$) is above a defined threshold α^{SINR} [70]. Therefore, to find handovers over the history it is necessary to analyse the SINR between each (user, ABS) pair at every simulation step. For example, a handover of user $u1$ from ABS 1 to ABS 2 happens when at step t, $\text{SINR}_{u1,1} > \alpha^{\text{SINR}}$ (the SINR between user $u1$ and ABS 1 is above the threshold), and then at step $t+x$ (where x is a certain time window, measured in a number of steps) we have that $\text{SINR}_{u1,2} > \alpha^{\text{SINR}}$ (the SINR between user $u1$ and ABS 2 is above the threshold) and also $\text{SINR}_{u1,2} > \text{SINR}_{u1,1}$ (user $u1$ is better connected to ABS 2 than to ABS 1).

Algorithm 1 Query to detect handovers. L is the current runtime log, T the set of timeslices in L, U the users, D the ABSes, $\text{SINR}_{u,d}(t)$ the link measurement between $u \in U$ and $d \in D$ at timeslice t, α^{SINR} the threshold for the SINR, and x a defined time window.

```
1:  Result = {}
2:  for each $u \in U$ do
3:    for each $d \in D$ do
4:      $T_B = \{t \in T | \text{SINR}_{u,d}(t) > \alpha^{\text{SINR}}\}$
5:      for each $t_b \in T_B$ do
6:        if $(\text{SINR}_{u,d}(t_b + x) < \text{SINR}_{u,d+1}(t_b + x) \wedge$
              $\text{SINR}_{u,d+1}(t_b + x) > \alpha^{\text{SINR}})$ then
7:          Add $(t_b, u, \text{SINR}_{u,d}(t_b), \text{SINR}_{u,d+1}(t_b + x))$
                to Result
8:        end if
9:      end for
10:   end for
11: end for
12: Result: Sequences showing handover transitions.
```

6 Evaluation of results

In this section, we present the evaluation and discussion of the results of using ETeMoX to explain the ABS SAS case study. We trained Q-Learning, SARSA and DQN under the same conditions. A training run consisted of 10 episodes and 2000 steps for 2 ABSes with 1050 users scattered on a X-Y plane. As mentioned, our implementation decouples the running RL-system from the generation of explanations. In that sense, the experiments were performed using two machines dedicated to different purposes: one performing the training of the different RL algorithms, and the other running ETeMoX. The RL algorithms ran on a virtual machine in the Google Cloud Platform[6]: specifically, an *a2-highgpu-1g* machine with 2vCPUs running Debian GNU/Linux 10 with 13GB RAM and an NVIDIA Tesla K80 GPU, using the ABS SAS simulator, Anaconda 4.8.5, matplotlib 3.3.4, numpy 1.19.1, paho-mqtt 1.5.0, pandas 1.1.3, and pytorch 1.7.1. The machine running ETeMoX was a Lenovo Thinkpad T480 with an Intel i7-8550U CPU with 1.80GHz, running Ubuntu 18.04.2 LTS and Oracle Java 1.8.0_201, using Paho MQTT 1.2.2, Eclipse Hawk 2.0.0, and Esper 8.0.0.

Our interest is to answer the stated RQs from Sect. 5. We analyse the costs of storing and retrieving explanations, as well as the accuracy of them on each experiment.

[6] https://cloud.google.com/.

6.1 Evaluation 1: evolution of a metric

Our architecture was able to sample the incoming data produced by the ABS SAS for each RL algorithm. Table 1 shows the costs of storing the TGDB using each approach. The full history of the system consisted of 40 000 model versions (10 episodes × 2 000 iterations × 2 ABSes). Depending on the sampling data rate selected, the size of the TGDB showed a linear decrease, going from approximate 130 MBs for the full history, to less than 1MB when sampling the history each 500 time steps.

The objective of this experiment was to keep track of the evolution of a metric throughout the history of the ABS SAS. For this case, it was the global reward: the number of connected users. In order to test the accuracy of the results, we ran a temporal query on the different TGDBs to find the averages from each training episode and see how they evolve. Figures 10, 11, 12 show the results. A t-test [64] was used to compare the means of each group. We compared the results using the full history to those from doing sampling at different rates. Table 2 shows the p-values for the null hypothesis H_0: there is no statistically significant difference between the sample sets. Anything with $p < 0.05$ is classed significant. Thus, we only reject the null hypothesis for the sample sets corresponding to the history sampled with data rate of 500. Therefore, they are significantly different from the base sample set (full history).

On the other hand, we also evaluated the costs for retrieving the information that built these visual explanations. Results are shown in Table 3. The query execution times also presented a linear decrease. Running the query in the TGDB corresponding to the full history took up to 43.23 s while running the query on the TGBD with the smaller size took between 0.08 and 0.09 s.

Table 1 TGDB size in MBs

Approach	Model versions	Q-Learning	SARSA	DQN
Full history	40000	126.00	129.00	162.00
History sampled r = 10	4000	15.00	16.00	24.00
History sampled r = 100	400	1.70	1.80	1.90
History sampled r = 500	80	0.95	0.46	0.77

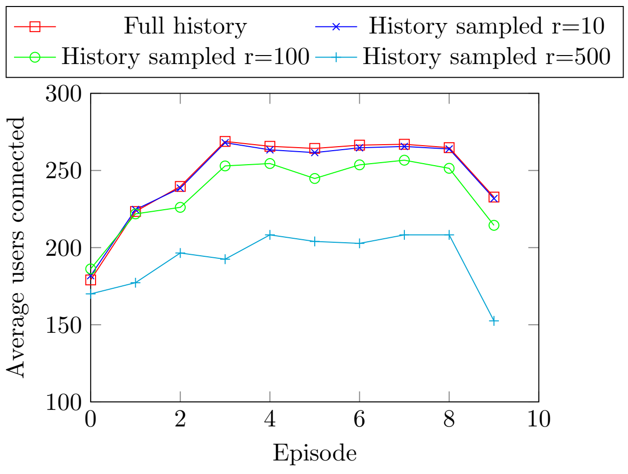


Fig. 10 **Q Learning:** Testing accuracy with data sampling approach

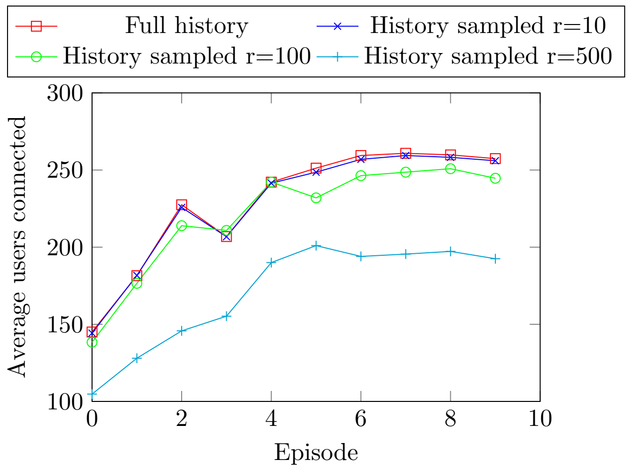


Fig. 11 **SARSA:** Testing accuracy with data sampling approach

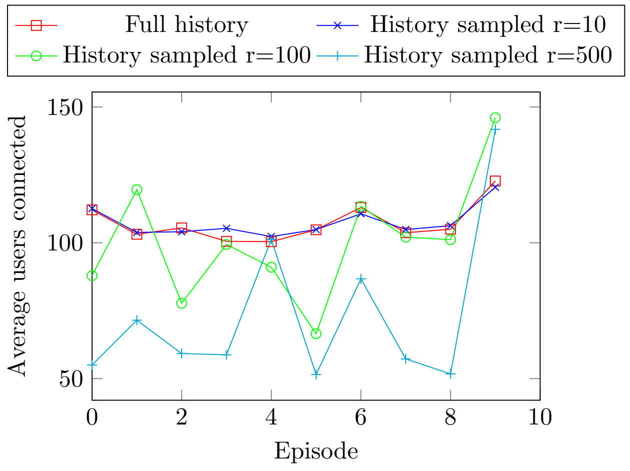


Fig. 12 **DQN:** Testing accuracy with data sampling approach

Table 2 T-test results

Approach	Q-Learning	SARSA	DQN
History sampled r = 10	0.95	0.94	0.88
History sampled r = 100	0.37	0.62	0.39
History sampled r = 500	1E-4	2E-3	5E-3

Table 3 Query execution times in Seconds

Approach	Model versions	Q-Learning	SARSA	DQN
Full history	40000	42.91	43.23	41.95
History sampled r = 10	4000	4.68	4.78	4.63
History sampled r = 100	400	0.34	0.34	0.38
History sampled r = 500	80	0.09	0.09	0.08

Table 4 Results and costs of filtering history with the exploration pattern

RL approach	Model versions	TGDB (MB)	Exploration (%)	Exploitation (%)
Q-Learning	562	8.80	1.41	98.60
SARSA	3195	18.00	7.99	92.01
DQN	3126	21.00	7.82	92.19

6.2 Evaluation 2: exploration versus exploitation

This experiment focused on finding situations where the action performed by the ABS SAS differs from the one that it currently thinks is best. An EPL query deployed in the CEP engine filters the history, letting through only the time steps where the ABS SAS was using exploration rather than exploitation. Table 4 shows a summary of the results of applying this filtering using the exploration EPL pattern. Both SARSA and DQN presented similar results, showing the system using exploration 8% of the time. In the case of Q-Learning, exploration was done during 1.41% of the time steps. The results of the EPL query selected the same time steps as a temporal query (EOL query) on the TGDB with the full history containing exploration events.

In order to compare the impact on accuracy of custom EPL-based filters in comparison with uniform sampling, we ran the same temporal query from Sect. 6.1 to find the reward averages for each episode on the different TGDBs for each RL-algorithm. Figure 13 shows the results for each approach. A similar behaviour to the one presented in the previous experiment is exhibited. Less data (model versions) create less precise results, as it is the case of Q-Learning. Although for SARSA and DQN similar number of model versions were found (3195 and 3126), the results show a significant variability for the case of DQN.

6.3 Evaluation 3: collaborations within the multi-agent system

The goal of this experiment was to prove the hypothesis of the developer that under ideal conditions and ideal communication between the ABSes, the system should reduce the number of handovers. We ran the temporal query on

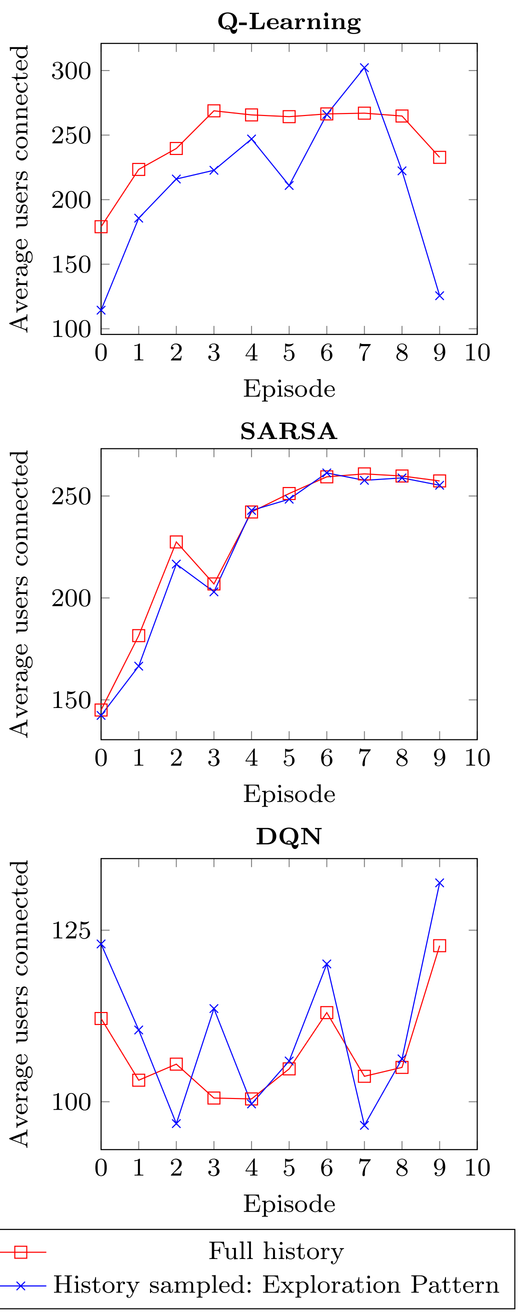


Fig. 13 Reward averages by episode on exploration pattern

the different TGDBs containing the full history. We used a SINR threshold of 40 ($\alpha^{\mathrm{SINR}} = 40$ in Algorithm 1), and a time window of 3 time steps ($x = 3$ in Algorithm 1) suggested by the developer to consider a transition time. The results were as follows: 1 784 handovers were found for Q-Learning, 590 for SARSA and 82 176 for DQN. An excerpt of the query results is presented in Listing 4. Line 2 indicates that a handover from ABS 1 to ABS 2 happened on SARSA on the episode 3 between time steps 912 and 915, when the user 477 was initially connected to ABS 1, and after

3 time steps was connected to ABS 2. A similar situation happened on line 4, but in this case there was a handover from ABS 2 to ABS 1 at episode 2 between time steps 1231 and 1234.

Due to the nature of the query, the execution times increased. They were as follows: 917s for Q-Learning, 1132s for SARSA and 7914s for DQN. This is because for each time slice (model version), it was needed to check how the SINRs for each user $u \in U$ changed over the defined time window. Thus, it was necessary to check across all 10 episodes (each spanning 2000 time steps) the SINRs for each of the 1050 users corresponding to each of the 2 ABSes. This produced $10 \times 2000 \times 2 \times 1050 = 42\,000\,000$ situations to check. A way to tackle the processing time can be using annotations as shown in [25]. Considering the previous, the situations found were very rare, representing only $4.2 \times 10^{-5}\%$ for Q-Learning, $1.4 \times 10^{-5}\%$ for SARSA. In DQN, although the situations still represented a very small percentage $1.9 \times 10^{-3}\%$, further studies about collaborative tasks are needed.

Listing 4 Excerpt of output from Algorithm 1 about handovers on the system's history (SARSA).

```
1  [ ... EolMap {SINR_1=80.76815264, ABS=1, user_id=477,
2  SINR_2=81.39525019, episode=3, step=912, step_handover
       =915}, ...
3  EolMap {SINR_2=80.52090196, ABS=2, user_id=925, SINR_1
       =81.20204141,
4  episode=2, step=1231, step_handover=1234}, ... ]
```

6.4 Answering the research questions

In Sect. 5, we presented four research questions. The answers to these questions are as follows:

- Answer to RQ1 on the feasibility of temporal models and complex event processing for XRL: the architecture of ETeMoX, powered by CEP and runtime models, provides an event-driven approach for TMs that can trace the evolution of the decision-making in RL-AI-models and the elements that affect it. The architecture can apply different filtering criteria to select relevant situations of the execution of RL systems. It enables an XRL technique that is independent of the RL-approach (i.e. AI-model-agnostic). ETeMoX is able to construct explanations that can focus on the evolution of specific metrics (Experiment 5.1), relate different metrics (Experiment 5.2) and also consider time to provide history-/time-aware explanations (Experiment 5.3).
- Answer to RQ2 on types of explanations: ETeMoX allows users to contrast the information of the different RL-algorithms through visual data-driven explanations. Plots, graphs and tables were used to illustrate the explanatory information, which will allow developers to control (i.e. debug) and discover insights from the system. Based on these explanations we suggest that the RL-algorithm with better performance for this case study is Q-Learning, while DQN shows the poorest performance and a variability on the collaborative task.
- Answer to RQ3 on costs: Regarding TGDBs sizes, when using sampling the different approaches showed a linear decrease. As expected, the TGDB corresponding to the full history required more disk space than the TGDBs with less model versions. Regarding the processing times, the behaviour was similar. Query processing times decreased with less model versions to visit. On the other hand, when using more complex filtering criteria (e.g. Experiment 5.2), the costs depend on the nature of the RL AI-model and the selection criteria.
- Answer to RQ4 on accuracy: Sampling the data at a rate of 10 time steps presented an accurate representation of the history, while requiring less resources (10% of the model versions): a t-test did not reveal statistically significant differences between using the full history and sampling at this rate. Based on the results, this allows for extracting similar conclusions from the sampled data. As an example, we can still conclude that the episode with greater reward is the 3rd on the case of Q-Learning, the 7th on SARSA and the 9th for DQN. Using a more complex filtering criteria for storing the history creates a TGDB of complex events with situations of interest for explanations.

6.5 Discussion

From the RL developer's point of view, retrieving historical information about the locations of the ABSes, their SINRs, and how many users were connected at specific time steps provided a better understanding of how ABSes interacted with the environment when using various RL-AI-models. The interaction data that was collected during training and execution retain more information than just the learned policy: studying how these metrics evolve reveals interesting challenges encountered by the ABSes.

By analysing the collaborations within the multi-agent system, the behaviour of each ABS gives us an understanding about the reasons why the ABSes with the Q-Learning RL-AI-model had more connected users overall (i.e. maximum global reward) than SARSA and DQN. The Q-Learning ABSes presented fewer handovers during the period of simulation compared to those using SARSA and DQN. Interestingly, the DQN ABSes had a similar proportion of exploration and exploitation to the SARSA ABSes. However, the DQN ABSes had far more handovers than the SARSA

ABSes. This allows developers to understand why the DQN ABSes performed worse than the SARSA ABSes in this case study.

Furthermore, the analysis of collaborations shows that SARSA-based ABSes have a level of knowledge of their neighbours' position and capacity. In this case study, an ABS is rewarded if it increases its number of connected users, even if it reduces the number of users of other ABSes. The total reward is not an implicit learning constraint for the ABSes. Experiment 5.3 showed, in both SARSA and Q-Learning cases, that the number of handovers that happened during the simulation was minimal. The latter seems to demonstrate that the ABSes were able to perceive the intention of neighbouring ABSes. In contrast, in the case of DQN, multiple situations were found that show violations of the collaboration principle. Therefore, more analysis on collaborative ABSes with DQN AI-model is needed to have better understand how Q-value function works in this case study.

Statistical and historical information can guide developers when working with AI-models to improve the learning performance: for example, by imposing a balance between exploration and exploitation if necessary. The exploration and exploitation query can give developers further insights about how to improve Q-Learning RL-AI-model performance by increasing the exploration time. We believe that this work can inspire promising RL researchers' frontiers to perceive the aptitude of agents driven by RL-AI-models with automatically generated behaviour observation through our framework.

6.6 Threats to validity

There are internal and external threats to validity. In terms of internal threats, queries ran could be formulated in several different ways, and a slightly different formulation could have produced different results. For instance, since the metamodel does not have an explicit attribute to mark exploration vs exploitation, then if having multiple actions associated with the best reward we could end up marking the action as using exploitation when it had not been the case. Similarly, defining a different time window for the handovers query would have produced different results. The shown queries and time windows were designed and configured in collaboration with the case study developers, and they encode their experience with RL: if needed, these can be modified without major changes to ETeMoX. Finally, we used the t-test assuming normality for comparing data sets, using different statistical approaches could have produced contrasting results. Such cases need to be studied further.

With respect to external threats, we developed the queries focusing on the needs and requirements of the developers of the ABS SAS. For other systems and other RL developers, other types of explanations may be needed. These new explanations would require new queries, which may impose new requirements on ETeMoX in terms of features or performance requirements. Another external threat is that while ETeMoX can now sample and filter the relevant history, still it does not allow the limitation of the recorded history to a given time window. Therefore, for longer training sessions, a capability for limitation as the one described may be needed for better management of resource consumption. Furthermore, if this temporal model were to live for long periods, it could undergo changes in its structure (i.e. by metamodel evolution). In other words, the temporal model would need to be made flexible enough to answer queries across the revisions implied.

7 Related work

7.1 Explainable and interpretable reinforcement Learning

RL AI-models has had great successes in solving multi-agent collaborative tasks when three conditions are met: having an environment, having a reward generation process, and having agents interacting with the environment. [67]. General RL algorithms require big quantities to obtain good performance based on interactions with the environment, which inhibits many practical applications, since obtaining environment interactions is often costly and challenging [44]. Having interpretable RL algorithms will allow developers to closely follow and fine turn the training process, improving the sampling efficiency of the RL process while reducing exploration time.

It is only recently that researchers have started investigating the way of explanations for RL algorithms. One approach is illustrating how the actions affect the total value of the policy [19,30]. Van der Waa et al. proposed allowing users to ask contrasting questions about why the RL algorithm followed a certain policy (the *fact*) instead of an alternate (*foil*) policy of their liking, having the RL algorithm trying to follow those actions as much as possible, while presenting explanations on how much the RL algorithm heeded those recommendations and also why it did deviate from them [73]. This approach is quite subjective, since users may have many different views.

Cashmore et al. also used contrasting questions to design an approach for explainable planning. It allowed developers to see the consequences of forcing a particular action to be taken (rather than the one suggested by the algorithm) [11]: their work mentioned the risks in improperly interpreting the question, and the difficulties in formalising questions about plan structures. Camacho et al. experimented with the use of high-level notations (automata-based representations known as *reward machines* or RMs) to reduce the burden of specifying reward functions, proposing translations from linear temporal logic formulas to RMs, and reporting improvements in sample efficiency [9].

The approaches more closely related to the one presented in this paper are reward decomposition [30] and interestingness elements [62]. Reward decomposition focuses on explaining why one action is preferred over another in terms of reward types (positive or negative). This analysis can be performed by our framework (with similar calculations), but it is not limited to rewards neither to a reward on a specific point in time. We can also present explanations about the system environment and states considering different time points and/or time windows. The interestingness elements approach collects data produced by the RL agent to present visual explanations. The authors conclude that the diversity of aspects captured by the different elements is crucial to help humans correctly understand an agent's strengths and limitations. Unfortunately, this approach lacks a formal structure to store the information, which is the advantage of using a MDE approach. Having this defined structure for representing decision traceability would allow algorithm developers to decouple the system to be explained from the rest of the explanation infrastructure. Once a translator from the logs to this metamodel is defined, the infrastructure and the queries that have been written against this metamodel can be used. The metamodel also enforces the use of a consistent level of abstraction across multiple algorithms in the same domain, helping perform comparisons among various RL algorithms. Moreover, our approach considers resource constraints, such as storage capacity and response times, tackling them through the use of TGDB and CEP, which is not part of [62].

7.2 Runtime monitoring

In this subsection, related work on runtime monitoring is classified depending on the approach in which it is based on, i.e. event-driven and CEP-based, graph-based monitoring, and runtime modelling.

In regard to the event-driven and CEP-based approaches for runtime monitoring, Fowler [23] proposed an event sourcing model that facilitates the traceability of the changes over time of the application state as an event sequence. However, event sourcing can be costly in terms of performance [45] since this model tracks every change leading up to a state. The present work describes an event-driven approach integrating CEP to both monitor event streams efficiently, and also deal with scalability problems through the use of temporal graphs for historical data.

Moser et al. [43] used CEP technology to create a flexible monitoring system with support for causal and temporal dependencies between messages for WS-BPEL service composition infrastructures. The proposal by Moser addresses several requirements: unobtrusive platform agnosticism, integration with other systems, multi-process monitoring, and anomaly detection.

Asim et al. [3] proposed an event-driven approach for monitoring both atomic and composite services at runtime. By using CEP, this approach allows for the real-time detection of contract violations. Additionally, Romano et al. [59] proposed the detection of contract violations through a quality of service (QoS) monitoring approach for cloud computing platforms. This approach integrated Content- Based Routing (CBR) with CEP.

Cicotti et al. [13] presented a cloud-based platform-as-a-service which is based on CEP and cloud computing. Service Level Agreements (SLAs) can be analysed by collecting Key Performance Indicators (KPIs) and defining CEP patterns. When KPIs exceed certain thresholds, the violation condition is prevented.

Barquero et al. presented an extension of CEP for graph-structured data in [4], where event streams are transformed into Spark datasets with a combination of persistent and transient data. The approach is able to take events from the Flickr and Twitter APIs, reshape them into graph datasets, to finally react on the fly to situations related to the interplay of these two social networks (using GraphX patterns). The authors reported good performance in these graph-structured scenarios, but it had worse performance than standard CEP engines in some situations, and writing SparkX queries proved to be difficult. In comparison, ETeMoX uses a standard CEP engine to filter what should be recorded into the TGDB: at the moment, reacting to events using graph-oriented queries would need periodic execution of a query in the provided temporal query language. Speeding up this type of periodic event detection query in ETeMoX would require the use of timeline annotation, which we have discussed in prior work [25].

With respect to work on runtime modelling, Gómez et al. [26] proposed the TemporalEMF temporal metamodelling framework, providing native temporal capabilities to models. This framework extends the Eclipse Modelling Framework (EMF) with the ability for model elements to store their histories in NoSQL databases, supporting temporal operators for retrieving the contents of the model at different points in time. On the other hand, TemporalEMF did not provide a full-featured temporal query language such as the Epsilon Object Language dialect in the Eclipse Hawk tool used in the present work.

Mazak et al. [39] proposed a runtime monitoring solution in which models can be partially mapped to time series databases. The solution is able to collect runtime information (i.e. time series data) and relate it to design models, ensuring traceability between design and runtime activities. More specifically, they presented a profile for annotating EMF metamodels with the ability to record the values of certain model element fields in time series databases, and query their historical information later.

In our most recent work [48], we conducted a feasibility study on the combination of temporal models and CEP for software monitoring. In particular, the proposed architecture was able to promptly respond to meaningful events (using CEP) as well as flexibly access relevant linked historical data (using TMs). In this previous work, CEP and TMs had not been integrated yet to help manage storage and processing trade-offs: they provided separate capabilities related to system history. The present work represents substantial progress in this area, as the CEP engine detects situation of interest that should be reflected in the TM, acting as a filter.

8 Conclusion and future work

This paper presented ETeMoX, an architecture based on Complex Event Processing (CEP) and temporal models (TMs) to support explainable reinforcement learning (XRL), targeting RL developers and RL-knowledgeable users. Explanations are generated using CEP and TMs, and presented visually through graphs and plots. ETeMoX has been applied to an RL-based self-adaptive system for mobile communications. In order to test the model-agnosticism of the approach, three different RL algorithms were used. The presented work has helped the case study developers to gain deeper insights about the behaviour shown by the running system and the reasons for its decisions. As such, the developers were able to obtain explanations about both the evolution of a metric and relationships between metrics. They were also able to track relevant situations of interest that spanned over time (i.e. over time windows).

To tackle the challenges in volume and throughput posed by data-intensive systems as in the case of RL, we used CEP as a real-time filter to select relevant points in time to be recorded in the TGDBs. Different filtering criteria were defined, and the trade-offs between storage and processing costs and the accuracy of the presented results were analysed. Uniformly sampling the history of the system every 10 time steps produced a good representation: a statistical t-test did not report significant differences in the query results compared to using the full history. This allows for extracting similar conclusions, while requiring less disk space (85% to 88% less) and taking less time to compute (88% less). Further, the system was able to correlate, process and filter data at runtime, providing the ability to flexibly define filtering criteria for building a TGDB of complex events. It allowed us to filter the history with domain-specific patterns in real-time to construct explanations about exploration vs exploitation. Finally, when using ETeMoX, the history can be studied back and forth, while looking for situations that took place over time. One example studied was handovers of users between collaborating airborne base stations, over certain time windows.

There are several avenues for future work. First, in relation to queries, processing times could be improved through the integration of timeline annotations [25], which allow the system to jump directly to situations of interest rather than scanning the full history of the TGDB. In terms of filtering criteria, CEP time window capabilities could be exploited for focusing on the last n versions to only keep a time window of the history in the TGDB, keeping resource consumption bounded.

For future work, explanations can play a key role for introducing the human-in-the-loop into RL-based systems. Presenting explanations at runtime and providing effectors/actuators for the user to interact with would allow the RL-developer to steer the learning process and modify it when required. This corresponds to level 3 of the proposed roadmap towards explainability in autonomous systems [49]. Additionally, distinct types of explanations can be targeted, such as global explanations to provide a general understanding of how the AI-model works. Further, the recipient of the explanations could be the system itself (i.e. *self-explanations*). With self-explanation support, the system could use its own history as another input to underpin its own decision-making (which corresponds to level 4 in the same roadmap [49]). Our technical future work will also include a benchmarking of ETeMoX with the proposed XRL approaches [30,62].

In addition, while the explanations were validated by the RL developers of the ABS SAS case study, further studies using other RL scenarios and developers about how explanations are understood are required in order to fully answer whether the explanations help RL developers in general to improve their systems. The type of explanations presented, either textual or graphical, make the targeted audience able to understand the data representations and can extract knowledge from their system. We plan to test the approach to evaluate the user perceptions using some well-known techniques for software acceptance as the Technology Acceptance Model (TAM) [17] and its variants [71]. Additionally, specific techniques to evaluate XAI as the one recently proposed by Rosenfeld in [60] will be explored.

Finally, we have focused on explanations for RL developers; however, there exist other stakeholders that can also be affected by AI-based systems. As mentioned in [29], there are different target audience profiles, each one with a different technical background. They include as follows: developers, experts (knowledgeable users), non-technical users, executives and regulatory agencies. Therefore, it is important to define the intended target audience and the pursued goal of the explanations generated (e.g. trustworthiness, informativeness, fairness, etc.). As such, more studies related to the explanation requirements from different actors in different scenarios are envisaged.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10270-021-00952-4.

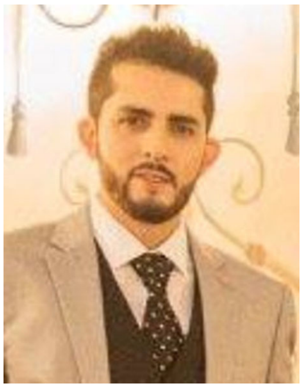
Acknowledgements This work has been partially sponsored by The Leverhulme Trust Fellowship "QuantUn: quantification of uncertainty using Bayesian surprises" (Grant No. RF-2019-548/9), the EPSRC Research Project Twenty20Insight (Grant No. EP/T017627/1), The Royal Society of Edinburgh project "A Reinforcement Learning Based Resource Management System for Long Term Care for Elderly People" (Grant No. 961_Yang), the Spanish Ministry of Science and Innovation and the European Regional Development Funds under project FAME (Grant No. RTI2018-093608-B-C33], and the Research Plan from the University of Cadiz and Grupo Ener-gético de Puerto Real S.A. under project GANGES (Grant No. IRTP03_UCA).

References

1. Adadi, A., Berrada, M.: Peeking inside the black-box: a survey on explainable artificial intelligence (xai). IEEE Access **6**, 52138–52160 (2018)
2. Anjomshoae, S., Najjar, A., Calvaresi, D., Främling, K.: Explainable agents and robots: results from a systematic literature review. In: 18th International conference on autonomous agents and multiagent systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019, pp. 1078–1088. International Foundation for Autonomous Agents and Multiagent Systems (2019)
3. Asim, M., Llewellyn-Jones, D., Lempereur, B., Zhou, B., Shi, Q., Merabti, M.: Event Driven Monitoring of Composite Services. In: 2013 International conference on social computing, pp. 550–557 (2013). https://doi.org/10.1109/SocialCom.2013.83
4. Barquero, G., Burgueño, L., Troya, J., Vallecillo, A.: Extending Complex Event Processing to Graph-structured Information. In: Proceedings of MoDELS 2018, pp. 166–175. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3239372.3239402
5. Bencomo, N., Götz, S., Song, H.: Models@run.time: a guided tour of the state-of-the-art and research challenges. Softw. Syst. Model. **18**(5), 3049–3082 (2019). https://doi.org/10.1007/s10270-018-00712-x
6. Blair, G., Bencomo, N., France, R.B.: Models@run.time. Computer **42**(10), 22–27 (2009). https://doi.org/10.1109/MC.2009.326
7. Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: MEdit4CEP: a model-driven solution for real-time decision making in SOA 2.0. Knowledge-Based Syst. **89**, 97–112 (2015). https://doi.org/10.1016/j.knosys.2015.06.021
8. Bucchiarone, A., Cabot, J., Paige, R.F., Pierantonio, A.: Grand challenges in model-driven engineering: an analysis of the state of the research. Softw. Syst. Model. **19**(1), 5–13 (2020)
9. Camacho, A., Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In: IJCAI **19**, 6065–6073 (2019)
10. Carey, P.: Data Protection: A Practical Guide To UK and EU Law. Oxford University Press Inc., Oxford (2018)
11. Cashmore, M., Collins, A., Krarup, B., Krivic, S., Magazzeni, D., Smith, D.: Towards explainable ai planning as a service. arXiv preprint arXiv:1908.05059 (2019)
12. Castelvecchi, D.: Can we open the black box of ai? Nat. News **538**(7623), 20 (2016)
13. Cicotti, G., Coppolino, L., Cristaldi, R., et al.: QoS Monitoring in a cloud services environment: The SRT-15 Approach. In: Euro-Par 2011: Parallel processing workshops. LNCS, pp. 15–24. Springer, Berlin, Heidelberg (2011)
14. Coggan, M.: Exploration and exploitation in reinforcement learning. CRA-W DMP Project at McGill University, Research supervised by Prof. Doina Precup (2004)
15. Corral-Plaza, D., Medina-Bulo, I., Ortiz, G., Boubeta-Puig, J.: A stream processing architecture for heterogeneous data sources in the Internet of Things. Comput. Standards Interfaces **70**, 103426 (2020). https://doi.org/10.1016/j.csi.2020.103426
16. Cox, M.T.: Metareasoning, monitoring, and self-explanation. Thinking about thinking, Metareasoning (2011)
17. Davis, F.D.: A technology acceptance model for empirically testing new end-user information systems: theory and results. Ph.D. thesis, Massachusetts Institute of Technology (1985)
18. De Freitas, E.P., Heimfarth, T., Netto, I.F., Lino, C.E., Pereira, C.E., Ferreira, A.M., Wagner, F.R., Larsson, T.: Uav relay network to support wsn connectivity. In: international congress on ultra modern telecommunications and control systems, pp. 309–314. IEEE (2010)
19. Dodson, T., Mattei, N., Guerin, J.T., Goldsmith, J.: An english-language argumentation interface for explanation generation with markov decision processes in the domain of academic advising. ACM Trans. Interact. Intell. Syst. **3**(3), 1–30 (2013)
20. Elastic: Introducting machine learning for the Elastic stack (2017). Last checked: 2020-05-15
21. Esling, P., Agon, C.: Time-series data mining. ACM Comput. Surv. **10**(1145/2379776), 2379788 (2012)
22. Feltrin, G., Popovic, N., Wojtera, M (2019) A sentinel node for event-driven structural monitoring of road bridges using wireless sensor networks. https://doi.org/10.1155/2019/8652527
23. Fowler, M.: Event sourcing. Online, Dec p. 18 (2005)
24. Fridman, L., Brown, D.E., Glazer, M., Angell, W., Dodd, S., Jenik, B., Terwilliger, J., Kindelsberger, J., Ding, L., Seaman, S., et al.: MIT autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation. arXiv preprint arXiv:1711.06976 **1** (2017)
25. Garcia-Dominguez, A., Bencomo, N., Parra-Ullauri, J.M., García-Paucar, L.H.: Querying and annotating model histories with time-aware patterns. In: 2019 ACM/IEEE 22nd International conference on model driven engineering languages and systems (MODELS), pp. 194–204. IEEE (2019)
26. Gómez, A., Cabot, J., Wimmer, M.: TemporalEMF: A Temporal Metamodeling Framework. In: J.C. Trujillo, K.C. Davis, X. Du, Z. Li, T.W. Ling, G. Li, M.L. Lee (eds.) Conceptual Modeling, Lecture Notes in Computer Science, pp. 365–381. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-00847-5_26
27. Haeusler, M., Trojer, T., Kessler, J., et al.: ChronoSphere: a graph-based EMF model repository for IT landscape models. Softw. Syst. Model. (2019)

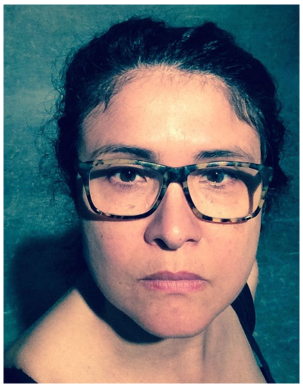
28. Hartmann, T., Fouquet, F., et al.: Analyzing complex data in motion at scale with temporal graphs. In: Proceedings of SEKE'17 (2017)
29. Heuillet, A., Couthouis, F., Díaz-Rodríguez, N.: Explainability in deep reinforcement learning. Knowl. Based Syst. **214**, 106685 (2021)
30. Juozapaitis, Z., Koul, A., Fern, A., Erwig, M., Doshi-Velez, F.: Explainable reinforcement learning via reward decomposition. In: IJCAI/ECAI Workshop on explainable artificial intelligence (2019)
31. Konno, S., Défago, X.: Approximate QoS Rule Derivation Based on Root Cause Analysis for Cloud Computing. In: 2019 IEEE 24th Pacific Rim international symposium on dependable computing (PRDC), pp. 33–3309 (2019). https://doi.org/10.1109/PRDC47002.2019.00020. ISSN: 2473-3105
32. Le Bras, P., Robb, D.A., Methven, T.S., Padilla, S., Chantler, M.J.: Improving user confidence in concept maps: exploring data driven explanations. In: Proceedings of CHI 2018. ACM (2018)
33. Letham, B., Rudin, C., McCormick, T.H., Madigan, D., et al.: Interpretable classifiers using rules and bayesian analysis: building a better stroke prediction model. Ann. Appl. Stat **9**(3), 1350–1371 (2015)
34. Light, R.: Mosquitto: server and client implementation of the MQTT protocol. J. Open Source Softw. **2**, 265 (2017)
35. Lim, B.Y., Dey, A.K., Avrahami, D.: Why and why not explanations improve the intelligibility of context-aware intelligent systems. In: Proceedings of CHI 2009. ACM (2009)
36. Lipton, Z.C.: The mythos of model interpretability: in machine learning, the concept of interpretability is both important and slippery. Queue **16**(3), 31–57 (2018)
37. Luckham, D.C., Frasca, B.: Complex event processing in distributed systems. Computer systems laboratory technical report CSL-TR-98-754. Stanford University, Stanford **28**, 16 (1998)
38. Mahmud, M., Kaiser, M.S., Hussain, A., Vassanelli, S.: Applications of deep learning and reinforcement learning to biological data. IEEE Trans. Neural Netw. Learn. Syst. **29**(6), 2063–2079 (2018). https://doi.org/10.1109/TNNLS.2018.2790388
39. Mazak, A., Wolny, S., Gómez, A., Cabot, J., Wimmer, M., Kappel, G.: Temporal models on time series databases. J. Object Technol. **19**(3), 3:1 (2020). https://doi.org/10.5381/jot.2020.19.3.a14
40. Merwaday, A., Guvenc, I.: Uav assisted heterogeneous networks for public safety communications. In: 2015 IEEE wireless communications and networking conference workshops (WCNCW), pp. 329–334. IEEE (2015)
41. Mitchell, R., Michalski, J., Carbonell, T.: An Artificial Intelligence Approach. Springer, Berlin (2013)
42. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
43. Moser, O., Rosenberg, F., Dustdar, S.: Event Driven Monitoring for Service Composition Infrastructures. In: L. Chen, P. Triantafillou, T. Suel (eds.) Web Information Systems Engineering—WISE 2010, Lecture Notes in Computer Science, pp. 38–51. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17616-6_6
44. Oh, J., Hessel, M., Czarnecki, W.M., Xu, Z., van Hasselt, H., Singh, S., Silver, D.: Discovering reinforcement learning algorithms. arXiv preprint arXiv:2007.08794 (2020)
45. Overeem, M., Spoor, M., Jansen, S.: The dark side of event sourcing: Managing data conversion. In: 2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER), pp. 193–204. IEEE (2017)
46. Panch, T., Szolovits, P., Atun, R.: Artificial intelligence, machine learning and health systems. J. Glob. Health **8**(2) (2018)
47. Parra-Ullauri, J.M.: ETeMoX event-driven temporal models for explanations (2021). https://gitlab.com/sea-aston/etemox
48. Parra-Ullauri, J.M., García-Domínguez, A., Boubeta-Puig, J., Bencomo, N., Ortiz, G.: Towards an architecture integrating complex event processing and temporal graphs for service monitoring (2021)
49. Parra-Ullauri, J.M., García-Domínguez, A., García-Paucar, L.H., Bencomo, N.: Temporal models for history-aware explainability. In: Proceedings of the 12th system analysis and modelling conference, pp. 155–164 (2020)
50. Paul, L.C.: Handoff/handover mechanism for mobility improvement in wireless communication. Glob. J. Res. Eng. (2014)
51. Petter, E.A., Gershman, S.J., Meck, W.H.: Integrating models of interval timing and reinforcement learning. Trends Cogn. Sci. **22**(10), 911–922 (2018)
52. Poole David Mackworth Alan, G.R.: Computational intelligence: a logical approach.(1998). Google scholar google scholar digital library digital library (1998)
53. Popovic, N., Feltrin, G., Jalsan, K.E., Wojtera, M.: Event-driven strain cycle monitoring of railway bridges using a wireless sensor network with sentinel nodes. Struct. Control. Health Monit. **24**(7), e1934 (2017). https://doi.org/10.1002/stc.1934
54. Puiutta, E., Veith, E.M.: Explainable reinforcement learning: a survey. In: International cross-domain conference for machine learning and knowledge extraction, pp. 77–95. Springer (2020)
55. Ras, G., van Gerven, M., Haselager, P.: Explanation methods in deep learning: users, values, concerns and challenges. In: Explainable and interpretable models in computer vision and machine learning, pp. 19–36. Springer (2018)
56. Ricciato, F., Widhalm, P., Craglia, M., Pantisano, F.: Estimating population density distribution from network-based mobile phone data. Publications Office of the European Union (2015)
57. Robinson, I., Webber, J., Eifrem, E.: Graph databases, second edn. O'Reilly (2015). ISBN 978-1-4919-3089-2
58. Roldán, J., Boubeta-Puig, J., Martínez, J.L., Ortiz, G.: Integrating complex event processing and machine learning: an intelligent architecture for detecting IoT security attacks. Expert Syst. Appl. (2020). https://doi.org/10.1016/j.eswa.2020.113251
59. Romano, L., De Mari, D., Jerzak, Z., Fetzer, C.: A novel approach to QoS Monitoring in the Cloud. In: 2011 First international conference on data compression, communications and processing, pp. 45–51 (2011)
60. Rosenfeld, A.: Better metrics for evaluating explainable artificial intelligence. In: Proceedings of the 20th international conference on autonomous agents and multiagent systems, pp. 45–50 (2021)
61. Samek, W., Wiegand, T., Müller, K.R.: Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. arXiv preprint arXiv:1708.08296 (2017)
62. Sequeira, P., Gervasio, M.: Interestingness elements for explainable reinforcement learning: understanding agents capabilities and limitations. Artif. Intell. **288**, 103367 (2020)
63. Spinner, T., Schlegel, U., Schäfer, H., El-Assady, M.: Explainer: a visual analytics framework for interactive and explainable machine learning. IEEE Trans. Visual Comput. Graphics **26**(1), 1064–1074 (2019)
64. Student: The probable error of a mean. Biometrika **6**(1), 1–25 (1908). https://doi.org/10.2307/2331554
65. Such, F.P., Madhavan, V., Liu, R., Wang, R., Castro, P.S., Li, Y., Zhi, J., Schubert, L., Bellemare, M.G., Clune, J., et al.: An Atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents. arXiv preprint arXiv:1812.07069 (2018)
66. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Advances in neural information processing systems, pp. 1038–1044 (1996)
67. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
68. Theodoridis, T., Solachidis, V., Dimitropoulos, K., Gymnopoulos, L., Daras, P.: A survey on ai nutrition recommender systems. In:

Proceedings of the 12th ACM International Conference on PErvasive technologies related to assistive environments, pp. 540–546 (2019)
69. Turilli, M., Floridi, L.: The ethics of information transparency. Ethics Inf. Technol. **11**(2), 105–112 (2009)
70. Valente Klaine, P.H.: Self-organization for 5g and beyond mobile networks using reinforcement learning. Ph.D. thesis, University of Glasgow (2019)
71. Venkatesh, V., Davis, F.D.: A theoretical extension of the technology acceptance model: four longitudinal field studies. Manage. Sci. **46**(2), 186–204 (2000)
72. Vijayakumar, K., Arun, C.: Automated risk identification using NLP in cloud based development environments. J. Ambient Intell. Hum. Comput. 1–13 (2017)
73. van der Waa, J., van Diggelen, J., Bosch, K.v.d., Neerincx, M.: Contrastive explanations for reinforcement learning in terms of expected consequences. arXiv preprint arXiv:1807.08706 (2018)
74. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)
75. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge (1989)
76. Zeng, Y., Zhang, R., Lim, T.J.: Wireless communications with unmanned aerial vehicles: opportunities and challenges. IEEE Commun. Mag. **54**(5), 36–42 (2016)
77. Zheng, C., Yang, S., Parra-Ullauri, J.M., Garcia-Dominguez, A., Bencomo, N.: Reward-reinforced generative adversarial networks for multi-agent systems. IEEE Trans. Emerg. Top. Comput. Intell. (2021)

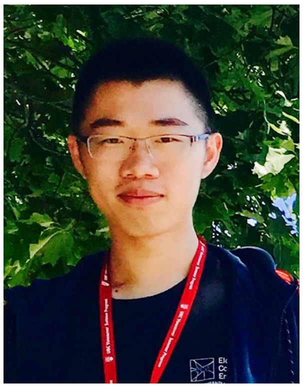


Juan Marcelo Parra-Ullauri obtained the degree of Engineer on Electronics and Telecommunications from the University of Cuenca, Ecuador in 2017. Currently, he is a PhD candidate at Aston University in the UK. His research interests include model-driven engineering, autonomous self-adaptive systems and explainability in AI-driven systems. Juan is interested in applying these approaches to cyber-physical systems, Internet of things and ambient assisted living contexts.

Antonio García-Domínguez is a Lecturer in Computer Science at Aston University (UK). His main research interests are software testing and model-driven engineering: in both of these fields, the increase in system sizes has required the adoption of AI-based approaches and non-relational database technologies to scale up. In addition to over 10 papers in peer-reviewed journals and over 40 papers in conferences and workshops, Antonio is a core contributor in several related open source projects. Some of these projects include the Eclipse Epsilon model management languages and tools, the MuBPEL mutation testing framework for WS-BPEL, or the Eclipse Hawk model indexing framework.



Nelly Bencomo is an Associate Professor at Durham University in the UK. Her research interests in Software Engineering include Decision-making under Uncertainty, Model-Driven Engineering, Autonomous Systems, Artificial Intelligence, and Requirements Engineering. She was awarded EU MC and UK Leverhulme Individual Fellowships. She is the PI of the EPSRC Project Twenty20 Insight. Contact her at nelly@acm.org or visit her web.

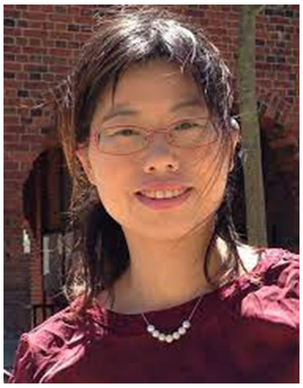


Changgang Zheng is a DPhil candidate at the University of Oxford and a member of Jesus College. He was graduated from the University of Electronic Science and Technology of China and the University of Glasgow joint school in 2020. Currently, he is a member of the Computing Infrastructure Group and working under the In-network machine learning project to explore the potential use of commodity programmable switches.

Chen Zhen is a master's student at the University of Science and Technology of China and he also graduated from the USTC. His interests include reinforcement learning and flight control of quadrotor unmanned aerial vehicle.

Juan Boubeta-Puig is an Associate Professor with the Department of Computer Science and Engineering at the University of Cádiz (UCA), Spain. He received his Ph.D. in Computer Science from UCA in 2014 and was honoured with the Extraordinary Ph.D. Award from UCA and the Best Ph.D. Thesis Award from the Spanish Society of Software Engineering and Software Development Technologies (SISTEDES). His research interests include real-time big data analytics through Complex Event Processing (CEP), Event-Driven Service-Oriented Architecture (SOA 2.0), Internet of Things (IoT), Blockchain and Model-Driven Development (MDD) of advanced user interfaces, and their application to e-health, smart city, industry 4.0 and cybersecurity.

Guadalupe Ortiz is Associate Professor in Computer Science and Engineering at the University of Cádiz. She has published over 100 peer-reviewed papers in international journals, workshops and conferences. She has participated in various programme and organisation committees of scientific workshops and conferences and acts as a reviewer for several journals. Her research interests embrace software architectures for context-aware services and their adaptation to edge devices, as well as the integration of CEP in service-oriented architectures in the scope of the IoT and sustainable smart cities.



Shufan Yang is the Associate Professor at Software Engineering Technology Subject Group, School of Computing at Edinburgh Napier University. Dr. Yang obtained her PhD degree in Computer Science from the University of Manchester in 2010. She is a chartered engineer and a fellow for British Computer Society and IEEE Senior Member. Dr. Shufan Yang's current research interests include cognitively inspired AI, reinforcement learning, deep learning, computer vision and AI+healthcare (ambient intelligent systems for healthcare). In the past she has also worked on system-on-chip and computational Neuromorphic computing.