Journal of Reliable Intelligent Environments (2024) 10:19–44
https://doi.org/10.1007/s40860-022-00198-x

ORIGINAL ARTICLE

Handling uncertainty in self-adaptive systems: an ontology-based reinforcement learning model

Saeedeh Ghanadbashi[1] · Zahra Safavifar[1] · Farshad Taebi[2] · Fatemeh Golpayegani[1]

Received: 18 June 2022 / Accepted: 20 December 2022 / Published online: 6 January 2023

Abstract
Ubiquitous and pervasive systems interact with each other and perform actions favoring the emergence of a global desired behavior. To function well, these systems need to be self-adaptive to handle noisy data and partially observable dynamic environments. However, facing unpredictable and rare events while only accessing incomplete information about the environment causes uncertainty in the adaptation process. Such uncertainty results in inconsistent decisions and unexpected system behavior. Currently, SAS handles such unpredictable conditions using adaptive modeling mechanisms to select default actions or exploiting reinforcement learning (RL) algorithms to learn new actions. However, the current mechanisms do not address rare events in an environment. This paper improves the system's decision-making when facing rare events by providing extra information about alternative adaptation actions using domain ontologies, which provide a thorough understanding of a domain. In this paper, we propose an Ontology-based unCertainty handling model (OnCertain), which enables the RL-based system to augment its observation and reason about the rare event using prior ontological knowledge. The overall aim of this model is to improve the system's decision-making process under conditions of uncertainty. Our model is evaluated in a traffic signal control system and an edge computing environment. The results show that the OnCertain model can improve the RL-based systems' observation and, consequently, their performance in such environments.

Keywords Self-adaptation · Uncertainty · Unanticipated change · Noisy sensor data · Reinforcement learning (RL) · Ontology · Traffic signal control · Edge computing

1 Introduction

Ubiquitous and pervasive computing aims to provide users with access to services all the time, everywhere, and transparently through the Self-Adaptive Systems (SASs) embedded in the physical environment. A SAS is a system that needs to adapt its run-time behavior autonomously by monitoring its environment and determining the best adaptation action to ensure that its adaptation goals are consistently achieved under run-time changing conditions [1,2]. For example, the Traffic Signal Control (TSC) system adapts its timing according to current traffic conditions [3], the task offloading algorithm adapts its strategy based on the task requests generated, their specific requirements, and available resources [4], and the home care management system adapts its way of communicating with the patients based on their physical and/or cognitive disabilities [5,6]. The environment is an external world where a SAS operates, comprising observable physical and virtual entities. The sensed environmental condition affects the decisions of the SAS, and these decisions can have new effects on the environment. The system's running environment gradually tends to be open, dynamic, complex, and noisy, making it difficult for developers to consider all possible changes [7]. This will cause the system to only deal with the predefined changes. So, there is a gap between a SAS's design and run-time models, which causes uncertainty in the adaptation process [8] and the system's behavior deviation from expectations [9].

To increase the effectiveness of systems' self-adaptation, it is essential having a mechanism to identify the source of uncertainty and mitigate it [8,10,11]. SASs would be able to alleviate at least some level of uncertainties using probabilistic run-time modeling mechanisms and optimization-driven

✉ Saeedeh Ghanadbashi
saeedeh.ghanadbashi@ucdconnect.ie

1 School of Computer Science, University College Dublin, Belfield, Dublin 4 D04 V1W8, Ireland

2 School of Computer Science, Islamic Azad University, Azarshahr Street, Tehran 1584743311, Iran

decision-making to select their default actions and exploiting machine learning (ML) and reinforcement learning (RL) algorithms to learn new actions [10]. Although the current approaches can model and reason about the uncertainty associated with a system and its environment [1,12], detecting sudden changes and handling open world new situations are their limitations [13]. Also, random action exploration may exhibit slow learning if there are many possible adaptation actions, and it will be more challenging when on the fly decision-making is desirable [1,14].

As a **motivating example**, in a TSC system, uncertainty can arise from fluctuations in travel demand due to weather conditions, commercial habits, and emergency events. If the TSC system encounters an ambulance entering one of its incoming roads, it must change the traffic signal phase from red to green to make clearance for the ambulance's paths. The system should select such a decision immediately so the ambulance can transport medical equipment, critical patients, and necessary medicines in time. Also, in the task offloading system, the rate of task requests generated by heterogeneous devices, different applications, and their specific requirements can be highly unpredictable. The task offloading service, which is faced with the sudden increase in workload, may adapt itself by rejecting lower priority task requests.

An ontology is a formal specification of a conceptualization of a domain [15], and it supplies a semantic description of data that a reasoning system can use to make new inferences [16,17]. In the literature, ontological knowledge is used to model systems and their environment [18–20]. However, the concerns related to handling uncertain environments, including detecting sudden changes and exploring the many possible adaptation actions, are not addressed.

In this paper, we propose a novel Ontology-based unCertainty handling model OnCertain that uses ontology and RL to better understand the system's environment and improve its decision-making. We focus on handling uncertainties caused by noise and unanticipated changes in the environment by adapting the sampling rate proportional to entities' importance, masking unnecessary parts of observation using entities' diverse characteristics, augmenting it using ontological knowledge, prioritizing the actions and their execution sequence using inference rules, and augmenting the reward signal using the actions' efficiency rate that improves its action selection. A TSC system and an Edge Computing (EC) environment evaluate the model's performance. The OnCertain model can enhance the performance of the SASs in such environments through improved observation.

The rest of this paper is organized as follows. Section 2 reviews relevant literature and Sect. 3 provides the required background knowledge. Section 4 briefly explains the problem statement. Section 5 describes our model and its different parts in the typical self-adaptation loop, which has to be extended to realize uncertainty handling using ontological knowledge. In Sect. 6, case studies are defined, and the results are analyzed. Finally, our conclusion and future works are drawn in Sect. 7.

2 Related work

To develop a system that is adaptive to an uncertain environment, various engineering approaches, such as eliciting adaptive requirements from the environment [21], analyzing SAS design while considering an uncertain environment [22], testing SAS implementation with environmental inputs [23], and updating environmental knowledge for optimal run-time decision-making [24], have been proposed. In [25], the authors specified loose self-management policies capable of flexible and nondeterministic choice of behaviors through the new Autonomic System Specification Language (ASSL). In [26], the authors proposed a goal model augmented with uncertainty annotations to guide the synthesis of adaptation policies at run-time. However, it is not specified how rare events such as catastrophic scenarios could be taken into account. Probability theory [27] is widely used to deal with different sources of uncertainty by running averages to mitigate uncertainty due to noise in monitoring, explicit annotation of adaptation strategies with probabilities, and estimation of the future environment and system behavior [28]. For instance, Rainbow framework [29] calculates the average of observations to deal with the uncertainty of the environment. Possibility theory has been mainly used in approaches that deal with the uncertainty of the objectives [30]. In [31], the authors designed a formal specification language RELAX to enable analysts to identify requirements that may be relaxed at run-time when the environment changes. The POISED approach was built on the possibility theory to assess the positive and negative consequences of uncertainty. It makes adaptation decisions that result in the best range of potential behavior, however, it can only deal with the defined uncertain situations [32]. Fuzzy control can tolerate the inaccuracy of sensors' data, but it requires the system engineers to extract fuzzy rules from historical data and cannot adapt to significant changes [33].

Artificial intelligence techniques, especially ML, have been adopted to deal with the difficulty in predicting environmental conditions at run-time [34]. ML can support the MAPE-K model [35] to build run-time models and adaptation strategies from complex and high-dimensional data obtained from uncertain environments [36]. For example, in [37], the authors used classifiers over the run-time data to detect real-time constraints. Then, these constraints are used to refine the design model of a SAS. In [38], the authors developed an ML-based approach, ACon, that uses MAPE-K feedback for adapting the SAS's contextual requirements affected by

uncertainty at run-time. A self-learning fuzzy neural network [33] was proposed to handle dynamic uncertainties from the sensing phase and deal with considerable changes. FUSION [27] adjusted strategies online through a learning model of the environment. However, ML-based approaches need to train the model before the system runs and cannot deal with the unknowns. Learning methods can handle this challenge by identifying the active context and modifying the adaptation space at run-time [39]. Thus, context changes not anticipated at design time are addressed by learning new adaptation rules dynamically or by modifying and improving existing rules [34]. Although, the generation of all possible situations, exclusively at run-time, poses a risk to the system's performance, reliability, and real-time constraints. In [14], the authors used the feature model for the system's adaptation space and thereby leveraged its semantics to guide RL's exploration, but their approach only supports discrete adaptation actions. Mao's team [40] proposed an RL method that can solve a part of unknown environment changes through existing strategies, however, they ignored that the existing strategies cannot solve all the novel situations. [41] proposed a planning method to handle uncertainty from the environment by learning knowledge of the relationship between system states and actions. This method can generate new strategies to deal with unknown situations, however, with the increase in system scale, more knowledge is needed to learn. When all possible changes in the environment are not known beforehand, and the models need to be derived during operation, RL techniques can be used [13], however, learning methods cannot handle unanticipated situations and detect sudden changes when they have not previously experienced such situations.

In the literature, several methods have been proposed to model and reason about the uncertainty, including formal specification languages, probability theory, ML, and learning techniques. However, it is not specified how rare/unknown events should be considered when they have not been experienced. In addition, exploring the significant number of possible adaptation actions may compromise the system's ability to make on the fly decisions under real-time constraints.

Ontology provides user-contributed, augmented intelligence, and machine-understandable semantics of data [16]. In SASs, ontology is mainly used as a modeling method to specify the requirements of a system-to-be. For example, in [18], the authors used a goal ontology containing semantic knowledge about the system's goals and the relationship between goals and sub-goals. In [42], to support decision-making in risk treatment, the authors designed a generic ontology for knowledge management in the MAPE-K loop to capture the expertise needed for various steps in safety management. In [19], the authors used a shared ontology, which includes concepts about the domain and conditions relating to them to support a shared understanding of concepts between the user and the system. The shared ontology enables the system to update the specification when a refinement or addition of new concepts occurs and also understand the service descriptions for solution provisioning. To manage the heterogeneity of real-world systems, in [43], the authors used ontologies, leveraging their ability to provide a shared and unified representation of a complex and heterogeneous domain of interest. In [20], the authors modeled the environment using an ontology that defines the concepts and available actions to the system for test generation. In [44], the authors proposed a framework to enable Continuous Adaptive Requirements Engineering (CARE) that leverages the requirements-aware systems and exploits a reasoning system. A fuzzy theory and semantic distance technology [45] were proposed to handle inaccurate monitoring data. Self-adaptive activity recognition systems [46] aim to recognize complex Activities of Daily Living (ADL) from a series of observations of the individuals' actions and the environmental conditions. In [47], the authors proposed a new approach that relies on an ontology to derive a first set of semantic correlation values between activities and sensor events. In [46], the authors introduced an incremental learning method to integrate new data with new features into current activity recognition models. In [48], the authors proposed a new domain adaptation technique that uses a general ontology to share and transfer activity models across individuals, even though the sensor deployments and operating environments are different. In [49], using an environment ontology, we proposed an Ontology-based Intelligent Traffic Signal Control (OITSC) model that enhances the RL controllers' observation and improves their action selection in dynamic and partially observable environments with stochastic traffic flow. However, in this paper, we propose that ontology can be used to address the uncertainty challenges of SASs in a different way. For example, the support of ontology to reason about unavailable data or extracting new concepts, relevant concepts to the actions, and the importance of concepts can be a great support.

3 Background

A brief overview of the required background information is provided in this section.

3.1 MAPE-K reference model

MAPE-K is a well-known reference model for SASs (see Fig. 1). MAPE-K's self-adaptation logic consists of four main conceptual activities: monitoring the system logic and the environment via sensors, analyzing the monitored data to determine whether adaptation is needed, planning adapta-

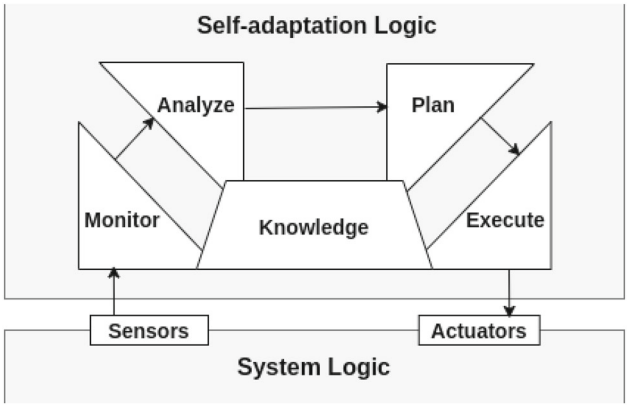


Fig. 1 MAPE-K reference model [35]

tion actions, and executing them via actuators modifying the system logic at run-time. Through the activities, a common knowledge base is used that contains information regarding the system and its environment, including models at run-time, adaptation goals, and strategies [50].

3.2 Partially observable Markov decision process

Formally, we describe a Partially Observable Markov Decision Process (POMDP) as a 7-tuple $(\Omega, S, A, R, T, V, \gamma)$, where S is a set of states ($s \in S$), A is a set of actions $A = \{a_1, a_2, \ldots, a_j\}$, T is the transition probabilities between states $T(s' \mid s, a)$ corresponds to the dynamics of the system, R is the reward function ($S \times A \rightarrow \mathbb{R}$), Ω is a set of observations $v \in \Omega$, V is the observation distribution given the state $v \sim V(s)$, and γ is a discount rate that shows how much the future reward signals contribute to the total reward signal. A policy is a function mapping the current state to the next action choice(s). It can be deterministic, $\pi : S \rightarrow A$ or stochastic $\pi : S \rightarrow \text{Prob}(A)$. The objective of the POMDP is to find an optimal policy π^* that maximizes the expected reward signal over the entire time horizon [51].

3.3 Reinforcement learning

RL is a trial-and-error method in which agents learn by interacting with the environment. For a SAS, agent refers to the self-adaptation logic of the system, and action refers to adaptation action. Reward signal gives feedback on the execution effect of adaptation action and adjusts decision-making. The agent interacts with the environment to get the observation $s_{g_i}^t$ and the reward signal r_i^t at time step t, and chooses the action a_i^t based on its policy. Finally, this action is executed in the environment simultaneously. The objective is to optimize the agent's expected reward signal. There are two types of RL algorithms: value-based (e.g., Q-learning, SARSA, DQN) and policy-based (e.g., DDPG). Q-learning algorithm approximates the discounted cumulative reward signal of actions Q-value (see (1)):

$$Q(s,a) = Q(s,a) + \rho[r + \gamma \max Q'(s',a') - Q(s,a)]. \quad (1)$$

$Q(s,a)$ is the current stored Q-value for applying action a in state s and max $Q'(s',a')$ is the maximum expected future reward signal. r is the reward signal value the agent gets from the environment after doing action a in state s, which takes the environment to the new state s'. The learning rate ρ determines to which degree the new information overrides the old one. SARSA algorithm learns Q-values relative to the current policy it follows. Deep Q-Network (DQN) algorithm uses a Deep Neural Network (DNN), and the input of the neural network would be the state that the agent is in, and the targets would be the Q-values of each of the actions. Deep Deterministic Policy Gradient (DDPG) combines both value-based (i.e., DQN) and policy-based (i.e., DPG) methods. Policy-based methods target modeling and optimizing the policy directly. RL-based agents choose between exploration and exploitation as a way to optimize the policy. The exploration scheme (exploring action space) and exploitation scheme (taking the best action) are combined to compute an optimal policy. ε is the percentage dedicated to the exploration [51].

3.4 Ontology

An ontology describes concepts, properties, relations, and axioms in a specific environment [15]. Using a hierarchical approach, ontology describes different relations between concepts such as binary, multiple, inverse, and conditional [52]. A relation can be used for a particular kind of instance (domain) with a particular value (range). A neighborhood concept set is a set of concepts with a direct relation (parents and children). A judgment is a statement asserting a certain relation for a concept. An inference rule is an implication of the form: If J_1, J_2 up to J_n are inferable, then J is inferable (see (2)). The inference rules are expressed using Semantic Web Rule Language (SWRL) [53]:

$$\frac{J_1, J_2, \ldots, J_n}{J}. \quad (2)$$

A concept weighting method is used to determine the weight of a concept [54], and the concept similarity measure is used to compare the similarities between the concepts in different ontologies [55]. The action ontology connects actions and concepts, thus forming a so-called "action environment" for each action. The action ontology is built to organize and structure action information [56], and its semantic constraints are typically defined as undesirable actions for the agent at each time step [57].

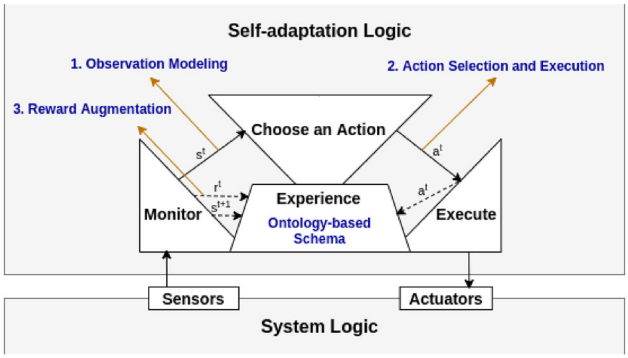


Fig. 2 An ontology-based uncertainty handling model (OnCertain)

4 Problem statement

The SAS considered in this paper includes uncertainties caused by noise and environment changes. To capture the underlying uncertainty and variability of dynamic environments, the knowledge model can be formulated as a stochastic dynamic decision problem generally modeled via a POMDP and can be solved using an RL framework. We assume that the observations received by agents are noisy. This type of uncertainty is caused by variation in a monitored agent parameter, changes in the process being monitored, or errors in the employed sensors, which results in different values for each observation taken. Also, when an agent's environment in which it executes changes, the characteristic of different entities in the agent's environment and the availability of different resources may change. This introduces uncertainty that may affect the quality of the agent's observations. So, there are N observations, i.e., $\Omega = \{v_{g_1}, \ldots, v_{g_N}\}$, given that a large majority of v_{g_i}s are uncorrelated with the corresponding s, i.e., they provide a poor representation of the current true state for the ith agent. This paper uses ontology to handle uncertainty due to noisy sensor data and environment changes.

5 Ontology-based uncertainty handling model

We present a new **On**tology-based un**Certain**ty handling model (**OnCertain**) that enables an agent to augment its observation and consequently improve its performance while face uncertainties. OnCertain introduces an RL-based MAPE-K model integrating RL and an ontology-based schema into a known MAPE-K loop. Figure 2 depicts the conceptual architecture of OnCertain. Using its sensors and actuators, the agent follows the process of monitoring, choosing an action, executing it, and updating its experience in the OnCertain model.

Using OnCertain, the agent accesses an ontology-based schema, a concept weighting method, a concept similarity measure, inference rules, and semantic constraints to augment its observations, reason about unanticipated changes in the environment, and improve its performance. Within this process, the agent models its observation by an ontology-based schema while monitoring its environment, selects and executes an action, and finally, computes a reward signal for that action. So, OnCertain includes the following stages: (1) observation modeling, (2) action selection and execution, and (3) reward augmentation (see Fig. 3). The details of these stages are discussed further in the following subsections.

5.1 Observation modeling

The agent's observation is represented using a schema described by an ontology. By this schema, the agent can interpret an unanticipated situation using a semantic description. The schema represents concepts perceived by the agent (i.e., monitored entities/parameters) and their relations. These relations enable inheritance between concepts and automated reasoning. We define $O^t_{g_i} = (C^t_{g_i}, P^t_{g_i}, M^t_{g_i})$ as the schema describing the data monitored/observed by agent g_i at time step t. $C^t_{g_i}$ represents the set of concepts $C^t_{g_i} = \{c_1, c_2, \ldots, c_x\}$, $P^t_{g_i}$ represents properties $P^t_{g_i} = \{p_1, p_2, \ldots, p_x\}$, and $M^t_{g_i}$ represents the set of relations over these concepts that expresses which concepts are associated with which concepts/values by which properties ($M \subseteq C \times P \times C$). Schema $O^t_{g_i}$ is used as input into all stages of the OnCertain model. Also, we model the action ontology taxonomy as $O_A = (C_{a_j}, P_{a_j}, M_{a_j})$, where C_{a_j} represents action-bound concepts, specifying which concepts are relevant to the action a_j, P_{a_j} represents properties, and M_{a_j} represents the relations between concepts and their properties.

In the **observation masking**, the agent focuses on important concepts/parts of the environment without distraction from irrelevant concepts/details and condenses broad sensory data into a compact form for action selection. Not seeing the irrelevant information helps the agent find similarities between its current and previous observations and use past experiences to respond to the current situation (i.e., generalization). During observation masking, similarities of the concept c_x in the ontology-based schema $O^t_{g_i}$ and the concept c_y in the action ontology O_A are compared. The similarity of the concepts c_x and c_y is measured based on their neighborhood concept set and the property set in the two different ontologies (see (3)):

Fig. 3 The OnCertain model's state diagram

$$
\begin{aligned}
&\mathrm{Sim}(c_x, c_y) = \sqrt{\frac{\alpha\,\mathrm{Sim}_C^2 + \beta\,\mathrm{Sim}_P^2}{2}},\\
&\alpha = \frac{|N_{g_i}^t(c_x)|}{|N_{g_i}^t(c_x)| + |N_{a_j}(c_y)|},\\
&\beta = \frac{|P_{g_i}^t(c_x)|}{|P_{g_i}^t(c_x)| + |P_{a_j}(c_y)|},
\end{aligned}
\tag{3}
$$

where $\mathrm{Sim}(c_x, c_y)$ is considered as the similarity measure between the concepts c_x and c_y and α and β specify the relative importance of the two similarity constraints Sim_C and Sim_P, respectively. Here c_x represents the selected concept of the agent's observation $v_{g_i}^t$ at time step t, c_y represents the concept that is relevant to the action a_j, $|N_{g_i}^t(c_x)|$ and $|P_{g_i}^t(c_x)|$ represent the number of neighborhood concepts and properties in the ontology-based schema and $|N_{a_j}(c_y)|$ and $|P_{a_j}(c_y)|$ represent that of the action ontology.

The function Sim_C is used for measuring the similarity of the neighborhood concept set of the selected concepts c_x and c_y in the two ontologies:

$$\mathrm{Sim}_C(c_x, c_y) = \sqrt{\frac{\frac{|N^t_{g_i}(c_x) \cap N_{a_j}(c_y)|}{|N^t_{g_i}(c_x)|} + \frac{|N^t_{g_i}(c_x) \cap N_{a_j}(c_y)|}{|N_{a_j}(c_y)|}}{2}}. \quad (4)$$

$|N^t_{g_i}(c_x) \cap N_{a_j}(c_y)|$ represents the number of common neighborhood concepts between the two ontologies. The function Sim_P is used for finding similarity by comparing property sets of the selected concepts c_x and c_y in both ontologies:

$$\mathrm{Sim}_P(c_x, c_y) = \sqrt{\frac{\frac{|P^t_{g_i}(c_x) \cap P_{a_j}(c_y)|}{|P^t_{g_i}(c_x)|} + \frac{|P^t_{g_i}(c_x) \cap P_{a_j}(c_y)|}{|P_{a_j}(c_y)|}}{2}}. \quad (5)$$

$|P^t_{g_i}(c_x) \cap P_{a_j}(c_y)|$ represents the number of common properties between the two ontologies. If $\forall c_y \in C_{a_j} : \mathrm{Sim}(c_x, c_y) < \text{threshold}$, then the agent does not use the observed data of the concept c_x in its decision-making for the action a_j (see lines 2–13 of Algorithm 1).

Algorithm 1 Observation-Modeling($v^t_{g_i}$, A, $O^t_{g_i}$, O_A)

```
1: for c_x in C^t_{g_i} do
2:   for a_j in A do
3:     f = 1
4:     for c_y in C_{a_j} do
5:       Compute Sim(c_x, c_y)
6:       if Sim(c_x, c_y) >= threshold then
7:         f = 0
8:       end if
9:     end for
10:    if f == 1 then
11:      Mask c_x for a_j
12:    end if
13:  end for
14:  if p_x has unknown value then
15:    for c_y in C^t_{g_i} do
16:      Extract judgments J_1, J_2, ..., J_n related to c_x and c_y
17:      if New judgement J : c_x -p_x-> Y is inferable then
18:        Assign value Y to p_x
19:      end if
20:    end for
21:  end if
22:  for p_x in P_{g_i} do
23:    Extract judgment J : c_x -p_x-> Y
24:    if Y not in v^t_{g_i} then
25:      Augment v^t_{g_i} with Y
26:    end if
27:  end for
28: end for
29: for c_x, c_y in C^t_{g_i} do
30:   Sample c_x with rate ⌊iw(c_x) × u/n⌉
31:   Sample c_y with rate ⌊iw(c_y) × u/n⌉
32:   Create sampled observation v̄^t_{g_i}
33: end for
```

In the **observation augmentation**, the automatic inference mechanism can deduce the augmented observations for concepts that have unknown information, i.e., either there is no sensor to observe them, or there is a fault in the sensors, so the agent has not observed these concepts while they exist in the environment. Thus, applying inference rules $J_1 : c_x \xrightarrow{p_z} c_y$, $J_2 : c_y \xrightarrow{p_y} Y \Rightarrow J : c_x \xrightarrow{p_x} Y$ to the existing relation between c_x and c_y and the explicitly observed information p_y of concept c_y enables agent g_i to extract implicit observation data p_x of concept c_x (see lines 14–21 of Algorithm 1).

During **observation transformation**, agent g_i deduces new properties p_x of the concepts $c_x \in C^t_{g_i}$ from the domain ontology at each time step t. So, it can automatically generate a larger variety of training data under a broad range of low-level properties to feed into its RL algorithm. To do so, the agent investigates the properties P_{g_i} (i.e., property set in the domain ontology) and retrieves their domain and range, then augments its observation with the range of properties whose domain is a concept of the environment (see lines 22–27 of Algorithm 1).

The occurrence of an error in the observation data causes the RL-based agent to exhibit inconsistent behavior. In the **observation sampling**, to reduce the impact of monitored noisy sensor data on the action selection process of agent g_i, observation $v^t_{g_i}$ can be sampled [58]. Samples are typically subsets or extracts from the agent's observation. They are used when the agent cannot observe an environment completely (i.e., due to complexity or noise) [59,60]. The size of a sample influences the agent's action selection precision. Assuming that the sample size is constant, the agent needs to adjust/divide the sample size proportionally to the importance of the observed data to draw precise conclusions. An observation's importance is determined by the importance of the concepts $C^t_{g_i}$ involved in it. The concept weighting function proposed in [61] uses an iweighting indicator $iw(c_x)$ to quantify how important each concept $c_x \in C^t_{g_i}$ is in an environment (see (6)):

$$iw(c_x) = 1/|M^t_{g_i}(c_x)| \sum_{m \in M^t_{g_i}(c_x)}^{|M^t_{g_i}(c_x)|} iw(m_{xy}). \quad (6)$$

During the ontology development process, relations are manually weighted by ontology engineers and then the concept's weight $iw(c_x)$ is computed based on the average importance weights $iw(m_{xy})$ of the relations $m \in M^t_{g_i}$ of domain concept c_x restricted by their range c_y. Five degrees of weight are available for weighting relations based on their importance: "Lowest", "Low", "Middle", "High", and "Highest". These degrees can be converted to numerical values using predefined mappings.

Thus, agent g_i samples observation data $v^t_{g_i}$ at different sampling rates proportional to the weight of concepts $c_x \in C^t_{g_i}$. To do so, the monitor activity in the MAPE-K loop is

modified to allow the observation data of concept c_x with higher iweighting indicator $iw(c_x)$ to be sampled at a higher rate $\lfloor iw(c_x) \times u/n \rceil$ than observation data of concept c_y with lower importance weight $iw(c_y)$. The sampled observation $\overline{v_{g_i}^t}$ is formed as follows (see lines 29–33 of Algorithm 1):

$$\overline{v_{g_i}^t} = \{x_1, \ldots, x_{\lfloor iw(c_x)\times u/n \rceil}, y_1, \ldots, y_{\lfloor iw(c_y)\times u/n \rceil}\} \quad \text{if } iw(c_x) + iw(c_y) = n \text{ and sample size} = u. \tag{7}$$

5.2 Action selection and execution

Typically, the agent must select its actions in dynamic and unpredictable environments, and specific actions are unavailable at every time step. Also, a fundamental challenge in RL is balancing exploration and exploitation of actions. Moreover, learning is concerned with identifying the optimal sequence of actions for the agent to execute to achieve the highest reward signal in the future. Uncertain situations can lead to challenges in finding the best sequence required to do so.

The agent's possible or valid actions are a small percentage of the available actions at each time step. The primary function of the **action masking** in RL is to filter out impossible or invalid actions, which improves its exploration performance by only considering valid actions and outputting a better policy. The agent's action selection in the current state is constrained by semantic constraints $H = (H^+, H^-)$ imposed by the ontology-based schema. An ontology language (e.g., SWRL) expresses them, specifying both the consequences H^+ that must hold in each state and the consequences H^- that cannot hold. These constraints are extracted from ontological domain knowledge. Any action that complies with these constraints is possible/valid and any action that violates these restrictions is impossible/invalid (see lines 1–9 of Algorithm 2).

In the **action exploration**, we propose an ontology-based policy to cope with the uncertainty of the environment. In this policy, the RL-based system dedicates a certain percentage of exploration time to the actions that reward them for the most important environmental concepts. The automatic inference mechanism deduces the appropriate action based on the ontology rules. Agent g_i employs a hybrid policy that combines the proposed ontology-based policy $\pi_{\text{OnCertain}}$ and a RL policy π_{RL}. It is also possible for g_i to switch between these two policies (see (8)), and $\alpha(t)$ is the percentage function to have a balance between them and is determined based on the distribution of unanticipated events experienced before time step t. Suppose unanticipated events occurred every 2 min, then $\alpha(t)$ will equal 0.5 (see lines 10–22 of Algorithm 2):

$$\pi(a \mid s) = \alpha(t) \times \pi_{\text{OnCertain}}(a \mid s) + (1 - \alpha(t)) \times \pi_{\text{RL}}(a \mid s). \tag{8}$$

Algorithm 2 Action-Selection-and-Execution($v_{g_i}^t$, A, $O_{g_i}^t$)

```
1: for a in A do
2:     for h in H do
3:         Check a comply with h
4:         if a not complied with h then
5:             Mask a
6:             break
7:         end if
8:     end for
9: end for
10: Generate random p1
11: if p1 <= α then                          ▷ The ontology-based policy
12:     Select c_x from C_{g_i}^t with the highest iw(c_x)
13:     Extract judgments J1, J2, ..., Jn related to c_x
14:     Infer a_i using J1, J2, ..., Jn
15: else                                     ▷ The RL policy
16:     Generate random p2
17:     if p2 <= ε then
18:         Select random a_j
19:     else
20:         Select a_k with the highest r_{g_i}^t
21:     end if
22: end if
23: for a in A do
24:     for j in J do
25:         Extract criterion b using j
26:     end for
27:     Compute action rank d for action a using b1, b2, ..., bn
28: end for
29: Select a_n with the highest rank d_{a_n}
30: Sort A' using ranks d_{a_1}, d_{a_2}, ..., d_{a_n}
31: Execute A' in order
```

In the **action prioritization**, actions are prioritized to address the requirements of ever-changing environments and limit the number of relevant actions in time-constraint situations. The RL-based agent computes rank d for the actions based on the criteria b proposed by the inference rules J. Then the computed rank is used for prioritizing the actions (see lines 23–29 of Algorithm 2).

In the **execution prioritization**, Algorithm 2 (see lines 30–31) uses action rank d to sort the sequence of actions $A' \subseteq A$ that should be executed in the environment.

5.3 Reward augmentation

The design of a proper reward can be challenging in complex problems, especially when there is uncertainty. This uncertainty may be due to environmental conditions that affect the sensory readings and need to be addressed to improve RL performance [62]. In the **reward augmentation**, the augmented reward signal $r' = r + f$ is proposed to provide more frequent feedback to the system, where r is the output of the original reward function R, f represents the additional reward signal resulting from a shaping function F, and r' represents the signal generated by the augmented reward function R' [63]. The proposed reward signal is augmented using the actions'

efficiency rate (see (9)):

$$r'_{g_i} = r^t_{g_i} + E. \tag{9}$$

Suppose we have a set of k actions $a_1, a_2, \ldots, a_k$. Each action changes n items while observing m items of state. For example, by observing the waiting time of m vehicles, the signal controller agent decides to switch to a green phase that causes a decrease in the waiting time of n vehicles. Let us consider an observation matrix $X = [x_{uv}, u = 1, 2, \ldots, m, v = 1, 2, \ldots, k]$ and a change matrix $Y = [y_{uv}, u = 1, 2, \ldots, n, v = 1, 2, \ldots, k]$. The qth line (i.e., X_q and Y_q) of these matrices shows quantified values of the observed concepts before applying action a_q and quantified values of the changed concepts after applying the action, respectively. We use the concept weights to compute the weighted sum of the quantified values. The efficiency rate (E) of the action is expressed as follows:

$$\begin{aligned} E &= \frac{\text{weighted sum of changed concepts' quantified values}}{\text{weighted sum of observed concepts' quantified values}} \\ &= \frac{\sum_{u=1}^{n} iw(c_u) \times y_{uq}}{\sum_{u=1}^{m} iw(c_u) \times x_{uq}}. \end{aligned} \tag{10}$$

The algorithmic procedure is shown in Algorithm 3.

Algorithm 3 Reward-Augmentation($v^t_{g_i}, v^{t+1}_{g_i}, r^t_{g_i}, O^t_{g_i}, O^{t+1}_{g_i}$)

```
for c_u in C^t_{g_i} do
    S_x += iw(c_u) × x_uq
end for
for c_u in C^{t+1}_{g_i} do
    S_y += iw(c_u) × y_uq
end for
E = S_y / S_x
r'_{g_i} = r^t_{g_i} + E
```

6 Performance evaluation

This section presents the evaluation scenarios and discusses the results.

6.1 Traffic signal control environment

Traffic Signal Control (TSC) system can adjust signal timing based on real-time traffic flow information to reduce traffic congestion. An Automatic TSC system can be managed by agents with learning capabilities to control traffic signals. This paper considers detectable and undetectable vehicles in the TSC environment. Communication devices are mounted on detectable vehicles to transmit real-time information concerning their waiting times and routes to the signal controller at an intersection. Undetectable vehicles do not possess such devices, and the signal controller does not have access to their information. To reduce the average waiting time of vehicles, we need to sample detectable vehicles' waiting time, augment the waiting time of undetectable vehicles, and explore actions regarding unforeseen situations due to dynamic and stochastic traffic flow behavior. To do so, the ontology developed in [49] (see Fig. 4) is used along with our proposed model.

6.1.1 Simulation settings

To evaluate the OnCertain model, SUMO (Simulation of Urban MObility) is used to simulate a 750 m × 750 m area with 16 intersections, each controlled by an intelligent entity, including two incoming and outgoing roads. Four types of vehicles, including default, ambulance, fuel truck, and trailer truck, with respective lengths of 5, 5, 10, and 10 m, are considered. One default vehicle enters the network from a random entrance point every second. Moreover, five important vehicles (ambulances, fuel trucks, and trailer trucks) enter the network every 2 min from random entrance points as unexpected/unforeseen events (i.e., we call this setting the base scenario). The minimum gap between any two vehicles is 2.5 m and the max speed allowed is 200 km/h, with the max acceleration of 2.6 m/s^2 and the decelerating of 4.5 m/s^2. The number of simulated seconds on SUMO is set to 1000 s.

6.1.2 Parameter modeling

The **state** is defined as traffic signal phases (i.e., yellow, red, and green), traffic signal phase elapsed time, the number of vehicles in each lane, the type of vehicles, and the vehicles' waiting time. The **action** is defined as selecting an appropriate traffic signal phase for the next time step. The **reward signal** is defined as the difference between the total waiting time of all vehicles in the current and next time step.

6.1.3 Scenarios

To evaluate the OnCertain model in situations with partial and noisy observations, we have defined two scenarios shown in Table 1.

The average waiting time of vehicles is used as a performance criterion. We compare the results obtained from the baseline RL algorithms, including Q-learning, SARSA, and DQN [64], to the same algorithms when they use OnCertain along with their usual functionality.

6.1.4 Results and discussion

The results report the average waiting time for all types of vehicles in 10 runs in each scenario (see Fig. 5). When

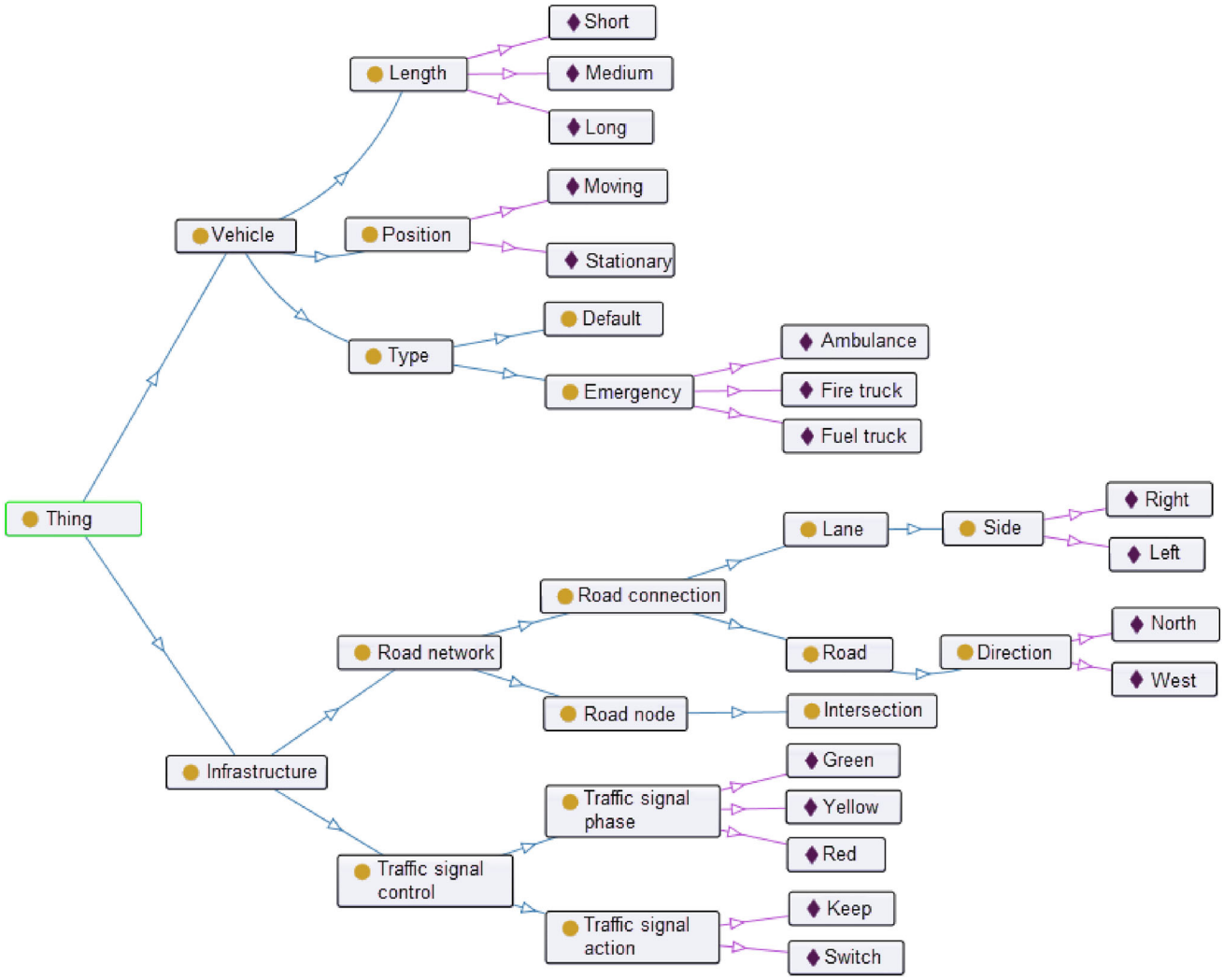


Fig. 4 Ontology for traffic signal control system [49]

Table 1 Traffic signal control environment—scenarios

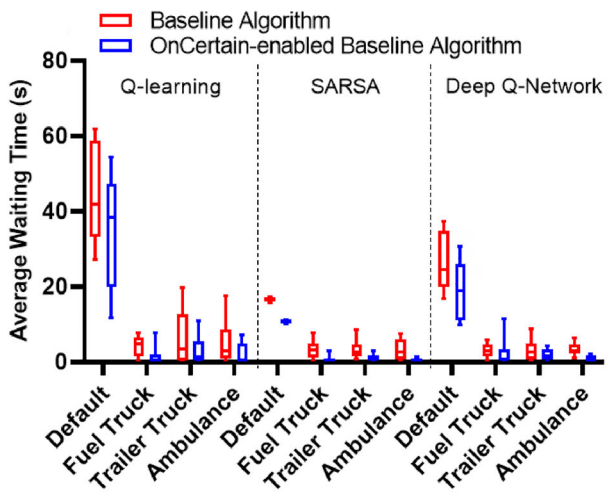
Scenario	Description
Base scenario with partial observation	20% of undetectable vehicles cannot be observed by the controller agent
Base scenario with noisy observation	50% of vehicles will have a corrupted waiting time

OnCertain is employed, the results show that all algorithms outperform their basic implementation.

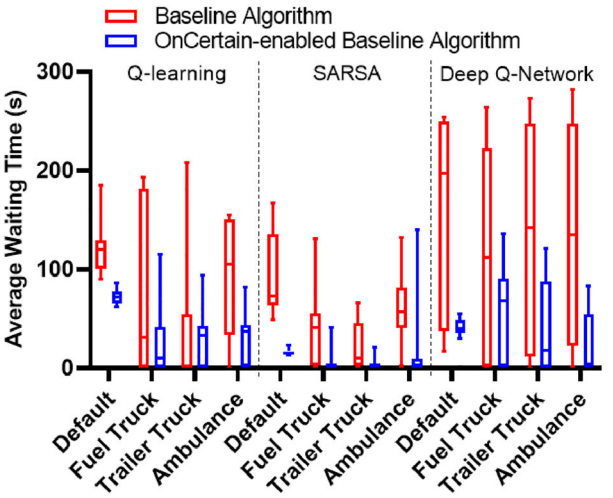
When important vehicles enter intersections, the controller assigns the "Highest" importance weight to the relation "hasType" and its domain "Vehicle" and range "Emergency", thus giving priority to the road that has the most important vehicles (see the related inference rules in Table 2). In this case, it has a higher efficiency rate to switch to a green phase for the road containing emergency vehicles than to keep a green phase on another road because the former action reduces the waiting time for important vehicles with the "Highest" importance weight.

When the OnCertain-RL signal controller is unable to collect data on undetectable vehicles in the environment, it replaces the waiting time of "Stationary" vehicles a on lane l with the current traffic signal phase elapsed time e, following the inference rule shown in Table 3.

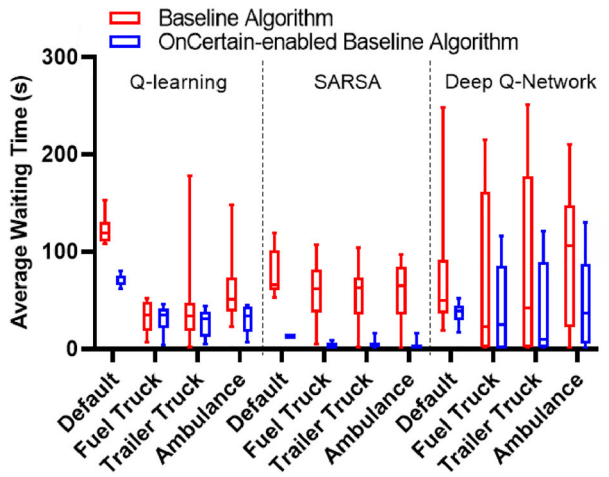
When the observation is noisy, the controller realizes that "Stationary" vehicles are more important than "Moving" vehicles according to the inference rules in Table 4, and their waiting times are sampled more frequently.



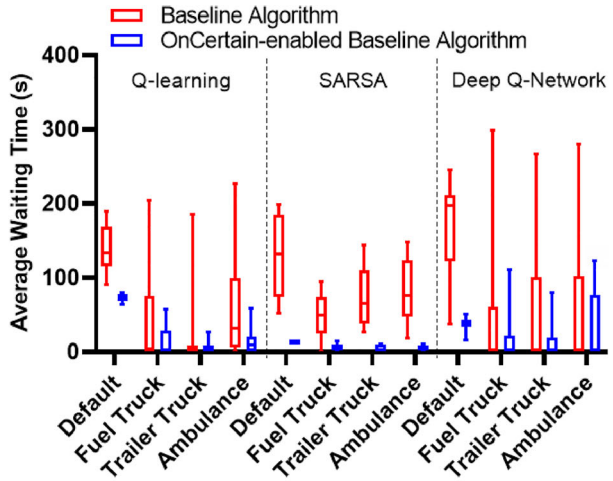
(a) Base scenario – Action exploration based on the vehicles' type.



(b) Base scenario with noisy observation – Observation sampling proportional to the vehicles' position.



(c) Base scenario with partial observation – Observation augmentation for waiting time of undetectable vehicles.



(d) Base scenario with partial observation – Reward augmentation for waiting time of detectable vehicles.

Fig. 5 The average waiting time of different types of vehicles—the OnCertain model and the three baselines

6.2 Edge computing environment

Edge Computing allows mobile devices to offload their latency-sensitive tasks to resource-rich edge servers. RL algorithms are used for task offloading, which requires considering all relevant information related to tasks and resources to make a reasonable decision. OnCertain can improve the RL algorithms' performance in several ways. From the various properties of tasks and resources fed into the RL algorithm, irrelevant observation data can be masked to address concerns such as learning speed, privacy, and limited bandwidth. Also, the RL agent can prioritize actions based on different criteria for actions, such as assigning tasks to servers with capacity limits. Moreover, to allocate resources for the tasks generated by latency-sensitive applications, there should be some semantic constraints/rules for prioritizing them at each step. To model the concepts in an

EC environment, we propose using the ontology shown in Fig. 6.

6.2.1 Simulation settings

To evaluate the performance of the OnCertain model in an EC environment, we simulated a neighborhood consisting of a varied number of users moving according to the mobility data collected from the users' devices at the subway station in Seoul, Korea [65]. People with smartphones, wearable gadgets, and laptops move around the neighborhood and use applications such as online games or voice over IP (VoIP). All these applications need considerable computational resources for execution. Users' devices offload tasks to one of 10 edge servers to obtain computation service. Edge servers are responsible for offering computational resources and processing tasks for mobile users. After a requested task has been processed, users need to receive the processed task from the edge server and offload a new task to an edge server again. Edge servers are connected with a mesh topology. The task may be migrated from one edge server to another within limited bandwidth. Information flow can only exist from one side to the other (i.e., the bandwidth is shared). All simulations are run during 25 steps; each of them takes 1500 s.[1]

6.2.2 Parameter modeling

Using the proposed EC ontology (see Fig. 6), the observation transformation defines the **state** as available computing resources of each edge server, its workload, board, group, cost, available migration bandwidth of each connection (link) between the servers, and task information including size, latency, priority (defined based on the application type, latency, or user profile), application type, user financial information (i.e., card number), user device information (i.e., location, type, operating system, and database), the offloading target of each mobile user, user group, and user usage history. The **action** is defined as computing resources each mobile user's task needs to use, migration bandwidth, and offloading target. The **reward signal** is defined as the total number of processed tasks in each step.

6.2.3 Scenarios

Our scenarios are designed to evaluate different steps of the proposed model, including observation masking, observation transformation, action masking, action prioritization, and execution prioritization in the simulated EC environment. These scenarios are as follows:

In the base scenario, we define that 80% of the servers can be used free of charge for task offloading, and the rest can be used at a cost. Also, 70% of the servers have a high board, and 30% of servers have a low board. 20% of servers can process four tasks simultaneously, 30% of servers process three, and 30% process two, and 20% of servers can only process one task simultaneously. Additionally, users and servers are organized into three groups based on user access type. In this setting, 30% of users are in group 1, 30% are in group 2, and 40% are in group 3. Also, 30% of servers are in group 1, 30% in group 2, and 40% in group 3.

This scenario includes heterogeneous tasks: (1) Health care tasks are: very low latency, small size, and high priority. (2) VoIP tasks are: very low latency, big size, and middle priority. (3) Entertainment tasks are: low latency, medium size, and low priority. (4) Data collection tasks are: high latency, very big size, and low priority. Also, this scenario creates fluctuated workload, where the number of users generating tasks in the environment and the rate of the latency-sensitive tasks are not constant throughout the simulation time (see Table 5).

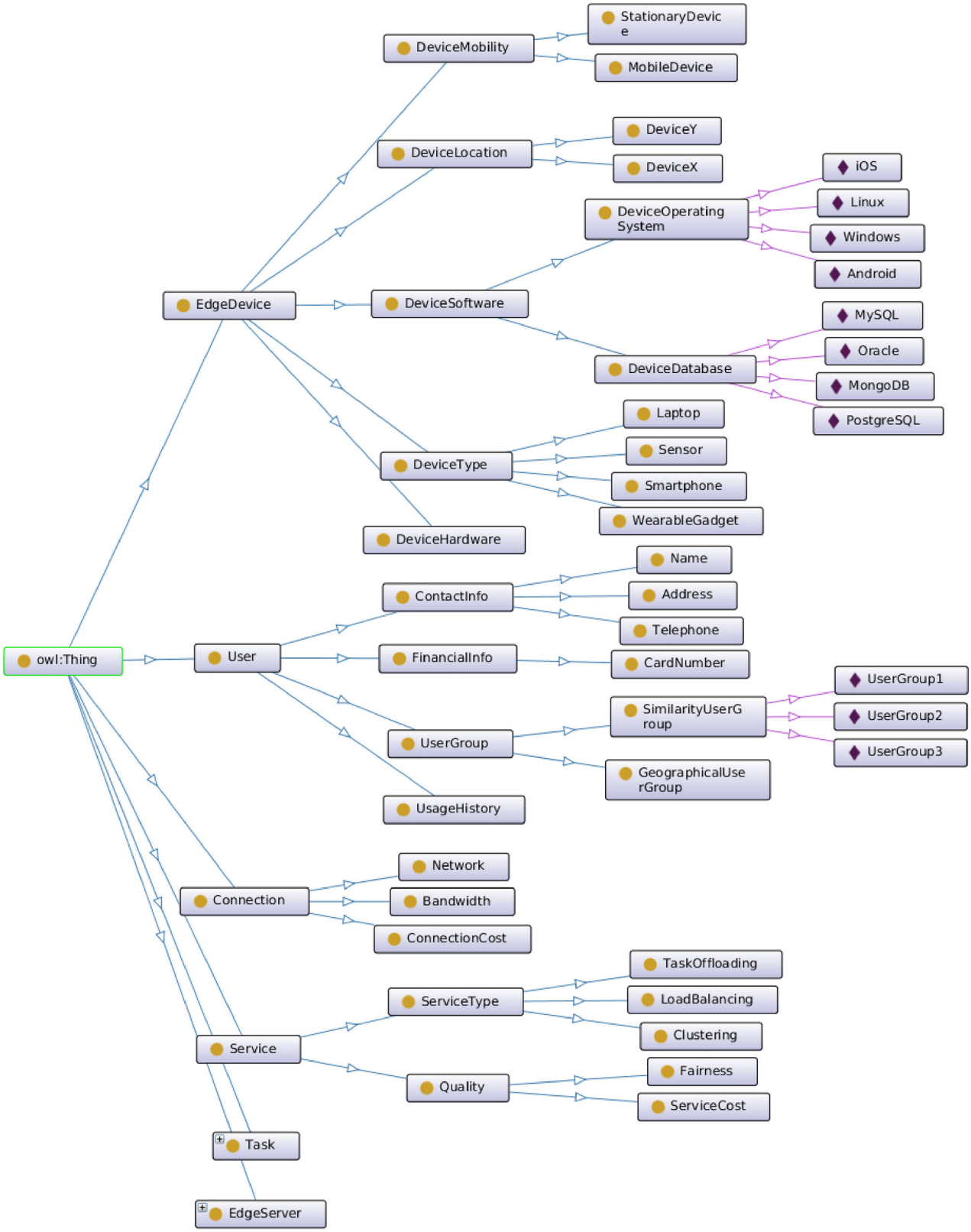
Evaluation metrics: Task success rate (i.e., average total processed tasks) is calculated by the number of processed tasks divided by the total number of generated tasks. The task failure rate (i.e., average total failed tasks) is calculated by dividing the number of tasks that failed due to the delay by the number of generated tasks. To evaluate the performance of the proposed model, we compare different task offloading strategies obtained from the baseline Deep Deterministic Policy Gradient (DDPG) algorithm [66] to the OnCertain-DDPG.

6.2.4 Results and discussion

Impact of Observation Masking: As described in the Sect. 6.2.2, the user device information, including its location, type, operating system, and database, is part of the observation. However, according to the EC ontology's inference rules shown in Table 6, only the device location must be kept, and the rest can be masked.

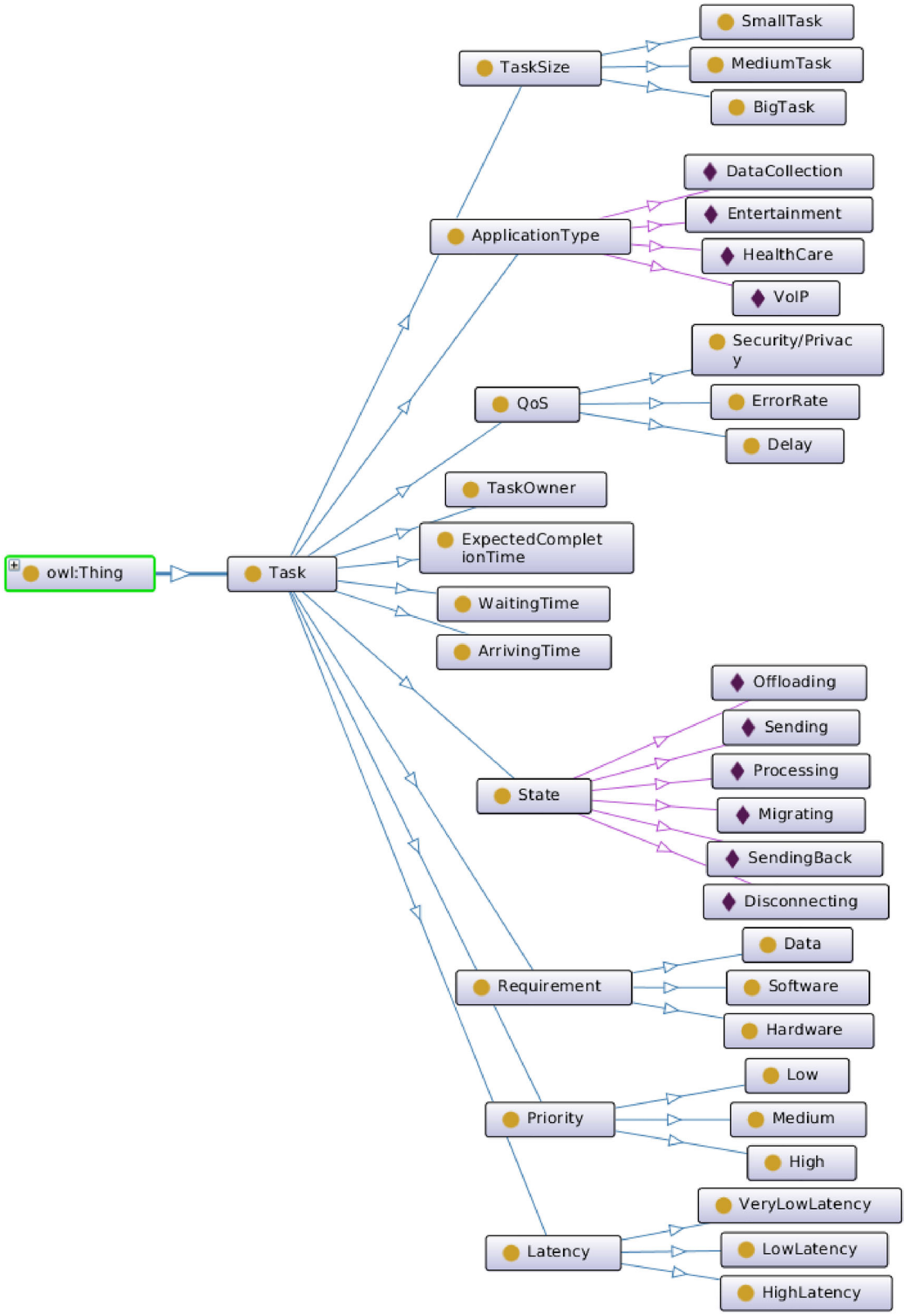
The average total processed and failed tasks in 10 runs in each scenario are reported in Fig. 7. The results show that the OnCertain-DDPG increases the average number of total processed tasks and decreases the average number of total failed tasks compared to the DDPG algorithm. We observe that masking users' device information as irrelevant information increases the average total processed tasks by 7%, 8%, and 7% in simple, medium, and hard scenarios, respectively. This improvement is 4%, 8%, and 9% for scenarios with 10, 25, and 50 users. The percentage increase is more significant when the number of users increases. The average total failed tasks is decreased by 6%, 3%, and 8% in simple, medium, and hard scenarios and 8%, 6%, and 3% in the scenarios with

[1] The code of OnCertain-DDPG algorithm and the simulated results are publicly available: https://github.com/saeedehghanadbashi/ontology-based-RL.



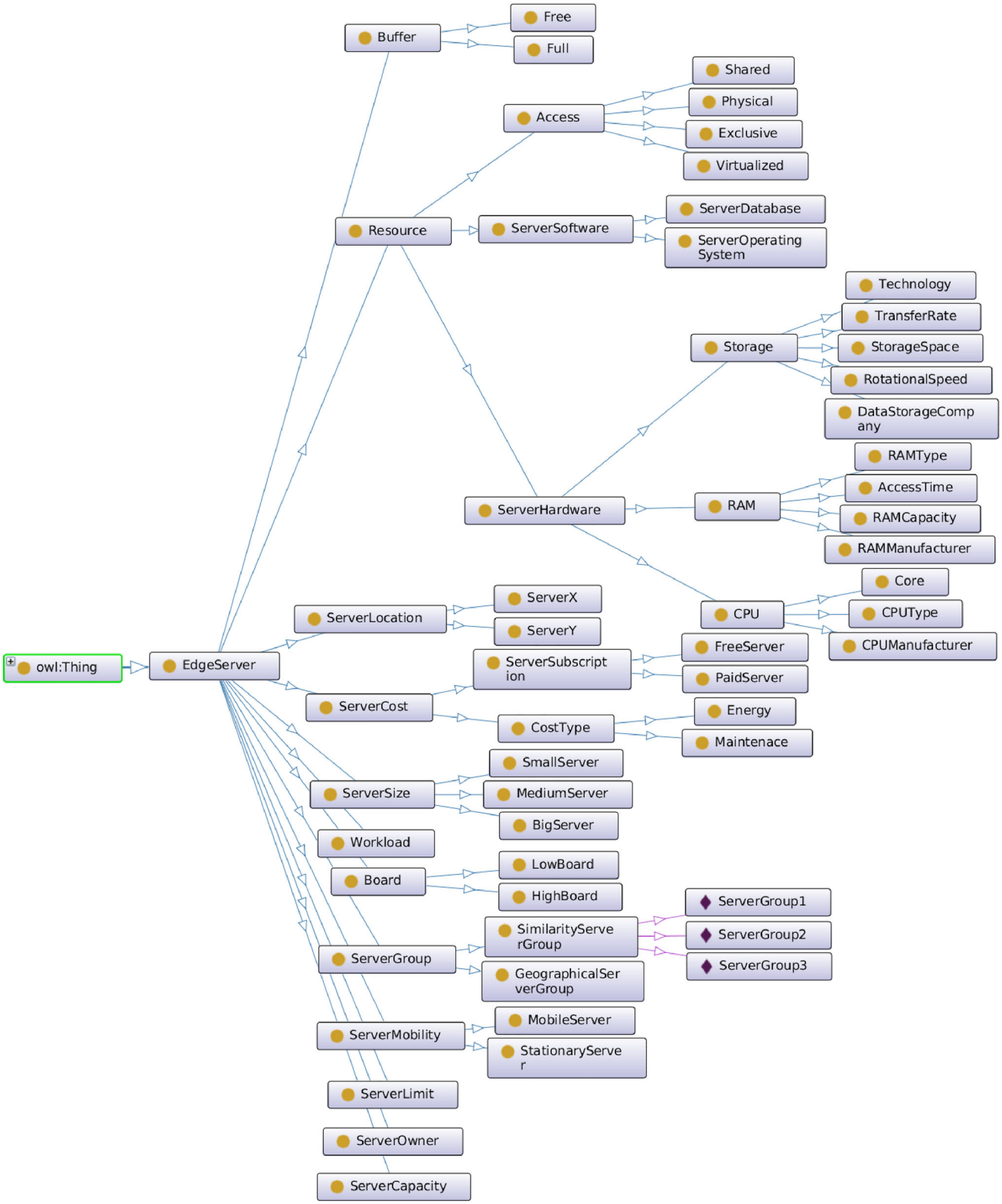
(a) Ontology for edge computing – Part 1.

Fig. 6 Ontology for edge computing



(b) Ontology for edge computing – Part 2.

Fig. 6 continued



(c) Ontology for edge computing – Part 3.

Fig. 6 continued

Table 2 Base scenario—action exploration based on the vehicles' type—an example of inference rules

Inference rule
TrafficSignalControl(?i[a]), Intersection(?s), Road(?r_1), Road(?r_2), Vehicle(?a), Vehicle(?b), isOn(?a, ?r_1), isOn(?b, ?r_2), atIntersection(?i, ?s), DifferentFrom(?r_1, ?r_2), isRegulatedBy(?r_1, ?i), isRegulatedBy(?r_2, ?i), hasType(?a, Emergency), hasSubType(?a, Ambulance), hasType(?b, Default), hasPosition(?a, Stationary), hasPosition(?b, Stationary), hasImportanceWeight(?a, Highest), hasImportanceWeight(?b, Low), hasTrafficSignalPhase(?r_1, Red), hasTrafficSignalPhase(?r_2, Green) => hasHigherImportance(?r_1, ?r_2), isNextTrafficSignalAction(?i, Switch)

[a]In Semantic Web Rule Language (SWRL), variables are indicated using the standard convention of prefixing them with a question mark

Table 3 Base scenario with partial observation—observation augmentation for waiting time of undetectable vehicles—an example of inference rules

Inference rule
TrafficSignalControl(?i), Intersection(?s), Road(?r), Lane(?l), consistOf(?r, ?l), Vehicle(?a), isOn(?a, ?l), atIntersection(?i, ?s), isRegulatedBy(?r, ?i), hasPosition(?a, Stationary), hasElapsedTime(?i, ?e) => hasWaitingTime(?a, ?e)

Table 4 Base scenario with noisy observation—observation sampling proportional to the vehicles' position—an example of inference rules

Inference rule
TrafficSignalControl(?i), Intersection(?s), Road(?r), Lane(?l), consistOf(?r, ?l), Vehicle(?a), Vehicle(?b), isOn(?a, ?l), isOn(?b, ?l), atIntersection(?i, ?s), isRegulatedBy(?r, ?i), hasPosition(?a, Stationary), hasPosition(?b, Moving), hasImportanceWeight(?a, Low), hasImportanceWeight(?b, Lowest) => hasHigherSamplingRate(?a, ?b)

Table 5 Edge computing environment—scenarios

Scenario	Workload	Latency-sensitive tasks
10-simple	10 users	5% of tasks are generated as health care, 15% of them are VoIP, 55% are data collection, and 25% are entertainment
25-simple	25 users	
50-simple	50 users	
10-medium	10 users	10% of tasks are generated as health care, 30% of them are VoIP, 35% are data collection, and 25% are entertainment
25-medium	25 users	
50-medium	50 users	
10-hard	10 users	20% of tasks are generated as health care, 40% of them are VoIP, 10% are data collection, and 30% are entertainment
25-hard	25 users	
50-hard	50 users	

10, 25, and 50 users, respectively. This indicates that there is a higher percentage decrease in the scenarios with less number of users.

According to the scenario setting discussed earlier, some edge servers can be accessed at a cost. In this case, users' payment information is required during the task offloading process. However, when using the free-of-charge servers, the users' payment information can be masked, according to the EC ontology's inference rules, as shown in Table 7.

We observe that masking users' card numbers as irrelevant information increases the average total processed tasks by 8%, 13%, and 4% in simple, medium, and hard scenarios, respectively. This improvement is 13%, 5%, and 7% for scenarios with 10, 25, and 50 users. The improvement is lower when the number of low-sensitive tasks increases. Also, the percentage increase is lower when the number of users increases. This is because the number of paid servers for which the user card number should be included in the

Table 6 Observation masking with user device information—an example of inference rules

Inference rule
EdgeServer(?s), User(?u), Task(?t), EdgeDevice(?d), isService(TaskOffloading), DeviceLocation(?l), DeviceOperatingSystem(?o), DeviceDatabase(?b), DeviceType(?v), hasEdgeDevice(?u, ?d), hasLocation(?d, ?l), hasType(?d, ?v), hasOperatingSystem(?d, ?o), hasDatabase(?d, ?b), hasTask(?u, ?t), requestFrom(?u, ?s) => isRelevant(?l), isIrrelevant(?o), isIrrelevant(?b), isIrrelevant(?v)

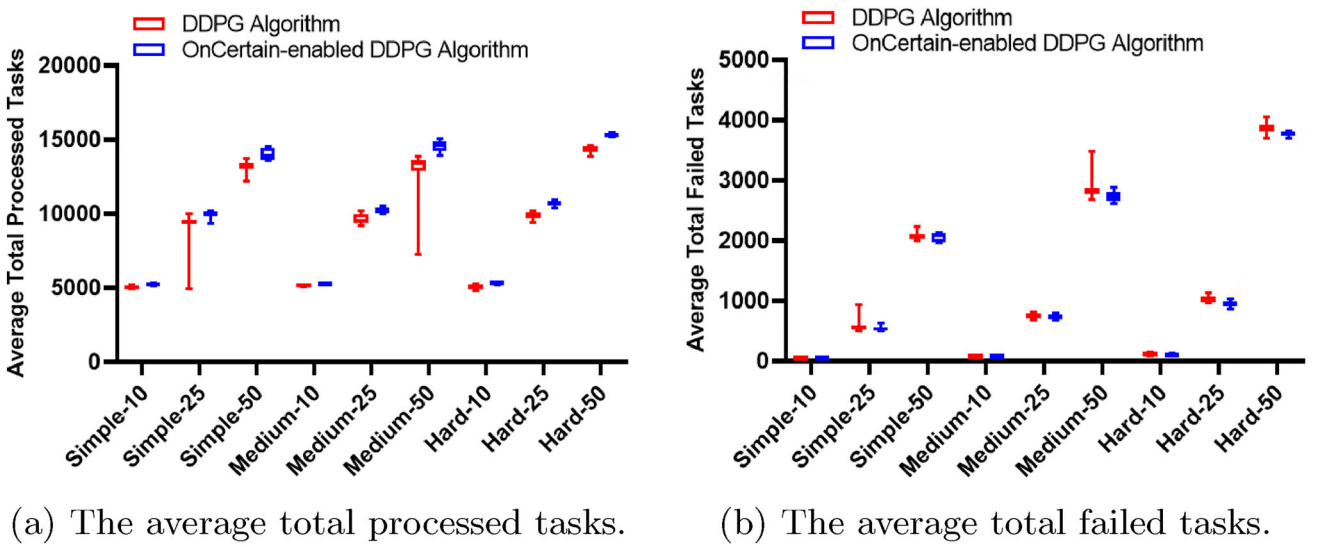


Fig. 7 Observation masking with user device information—the OnCertain model and the DDPG algorithm

Table 7 Observation masking with user card number—an example of inference rules

Inference rule
Server(?s), User(?u), Task(?t), CardNumber(?c), isService(TaskOffloading), hasCardNumber(?u, ?c), hasTask(?u, ?t), hasServerCost(?s, FreeServer), requestFrom(?u, ?s) => isIrrelevant(?c)

observation increases, and the observation in both methods will be similar. The same results are achieved in the comparison of the average total failed tasks so that it decreases by 8%, 9%, and 3% in simple, medium, and hard scenarios and 11%, 5%, and 5% in the scenarios with 10, 25, and 50 users, respectively (see Fig. 8).

Impact of action masking: Latency can be significantly reduced when the edge servers are closer to the place where data/task is generated. Using the inference rules of the proposed EC ontology (see Table 8), those servers far from the users are masked by giving the zero selection probability to the inaccessible servers, and the OnCertain-DDPG agent will not choose them.

The results show restricting the number of servers that can be assigned to each user based on the server board and user request latency requirement significantly increases the average total processed tasks and decreases the average total failed tasks compared to the baseline algorithm. The average total processed tasks increased by 18%, 16%, and 14% in simple, medium, and hard scenarios, respectively. This improvement is 23%, 16%, and 9% for scenarios with 10, 25, and 50 users. When the number of users increases, the number of available servers that can be assigned to each user based on the appropriate distance is limited, so less improvement can be the result. The same effects are observed in the percentage of decreases in the average total failed tasks by 32%, 29%, and 29% in simple, medium, and hard scenarios and 50%, 29%, and 11% in the scenarios with 10, 25, and 50 users, respectively (see Fig. 9).

Furthermore, considering servers' maximum capacity (server limit) is essential to balance the servers' workload. Using the predefined rules shown in Table 9, servers

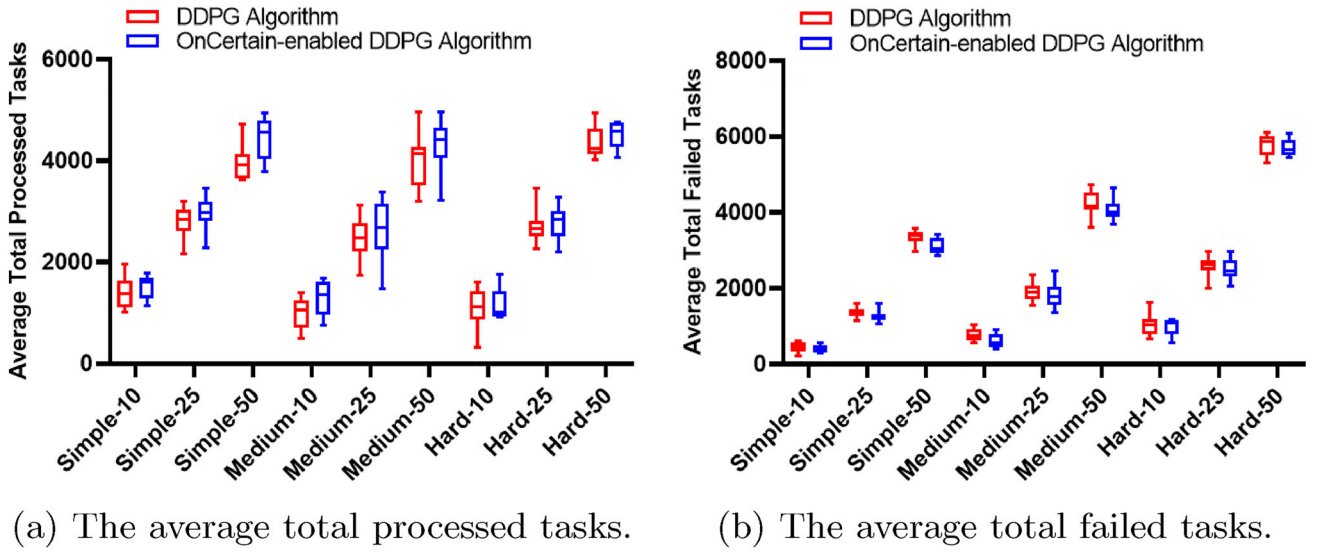


Fig. 8 Observation masking with user card number—the OnCertain model and the DDPG algorithm

Table 8 Action masking with server board—an example of inference rules

Inference rule
Server(?s), User(?u), Task(?t), ServerBoard(?b), Distance(?d), isService(TaskOffloading), hasServerBoard(?s, ?b), hasTask(?u, ?t), requestFrom(?u, ?s), hasDistance(?u, ?s, ?d), isGreaterThan(?d, ?b) => isInaccessibleFor(?s, ?u)

with computationally intensive workloads (i.e., unavailable servers) can be masked. This prevents servers from becoming overworked, which could cause them to slow down, drop requests, and even crash.

In Fig. 10, it is shown that masking the action set based on the server limit increases the average total processed tasks by 9%, 8%, and 7% in simple, medium, and hard scenarios and 8%, 8%, and 7% for scenarios with 10, 25, and 50 users, respectively. Also, the proposed model significantly decreases the average total failed tasks by 77%, 69%, and 72% in simple, medium, and hard scenarios and 88%, 83%, and 47% in the scenarios with 10, 25, and 50 users, respectively. The reason for the significant improvement in the average total failed tasks is that when load balancing is considered, it will prevent a server from saturating more than its limit, thus failing the tasks.

Impact of Action Prioritization: Edge devices can be grouped based on user access type before getting access to the servers of the EC environment to satisfy security requirements. According to the inference rules shown in Table 10, the action prioritization step prioritizes users' access to the servers in the same group.

The results indicate that prioritizing the servers for each user based on server group significantly increases the average total processed tasks satisfying security requirements compared to the baseline algorithm. The percentage increase is 82%, 78%, and 84% in simple, medium, and hard scenarios and 78%, 83%, and 83% for scenarios with 10, 25, and 50 users, respectively (see Fig. 11a). The average total processed tasks and average total failed tasks are not compromised to increase the satisfying security requirements (see Fig. 11b, c). This is a significant result that OnCertain improves satisfying security requirements while maintaining overall system performance.

Impact of Execution Prioritization: In the EC environment, since not all task requests from edge servers can be scheduled on time, thus, guaranteeing fairness among the users (i.e., edge devices offloading tasks) while considering the priorities of the tasks becomes a critical issue. Based on the inference rules, the OnCertain model proposes four execution prioritization strategies, namely the user fair, priority fair, application type fair, and latency fair, that account for the user usage history (i.e., the number of user tasks that have been processed), task priorities, application types, and task latency requirements, respectively (see Tables 11, 12, 13 and 14). So, the tasks are first prioritized, and then the servers are assigned to them.

The results indicate that prioritizing the execution of the actions based on usage history significantly decreases the average standard deviation of usage history compared to the

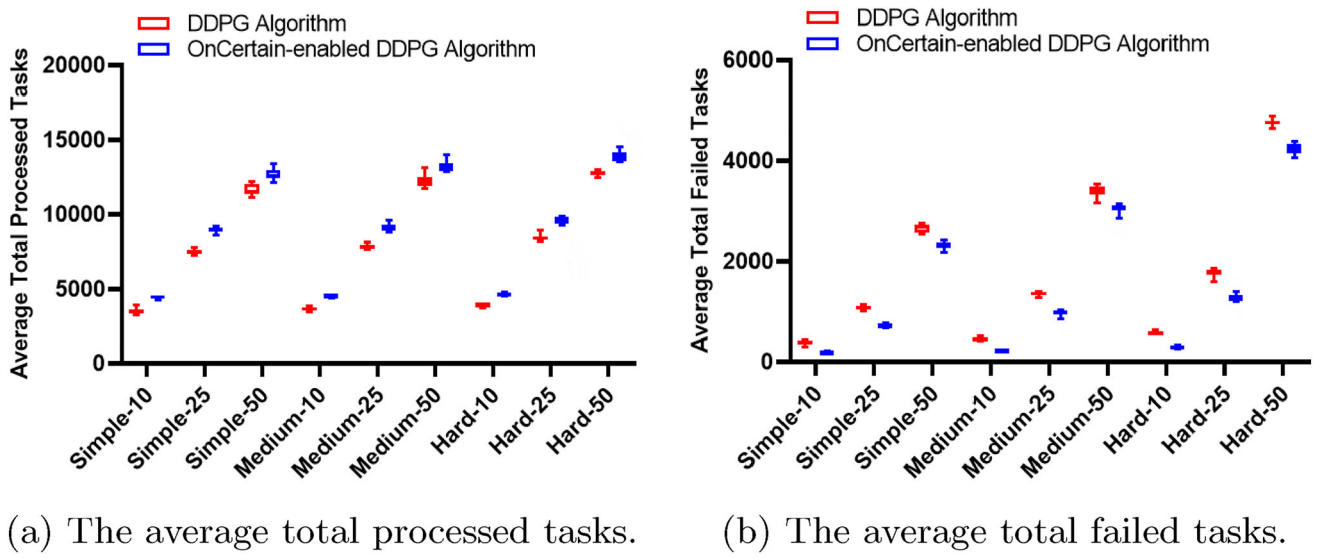


Fig. 9 Action masking with server board—the OnCertain model and the DDPG algorithm

Table 9 Action masking with server limit—an example of inference rules

Inference rule
Server(?s), User(?u), Task(?t), ServerLimit(?l), ServerWorkload(?w), isService(TaskOffloading), hasServerLimit(?s, ?l), hasServerWorkload(?s, ?w), hasTask(?u, ?t), requestFrom(?u, ?s), isEqualTo(?l, ?w) => isUnavailableFor(?s, ?u)

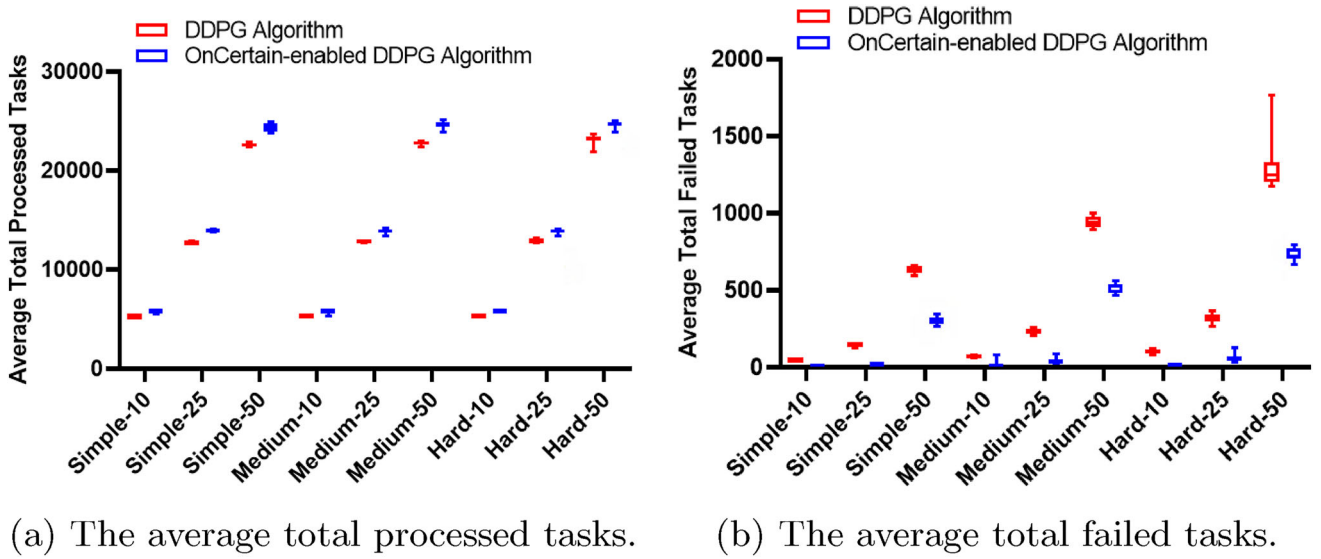
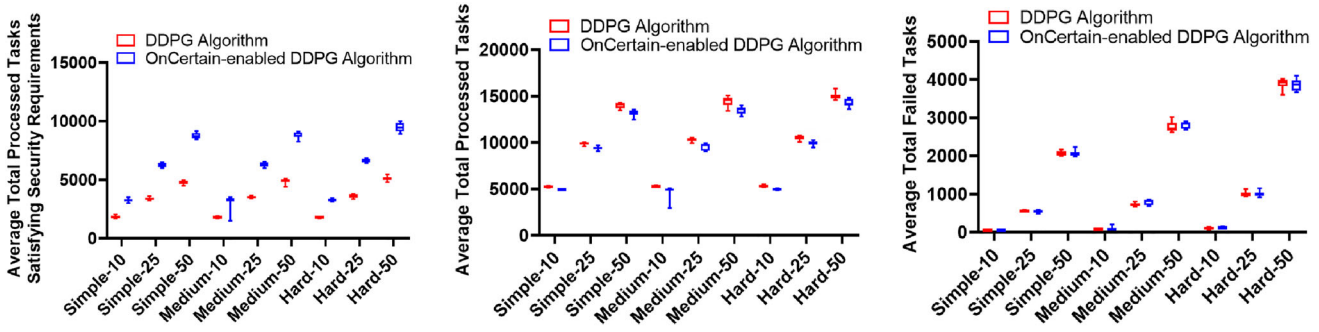


Fig. 10 Action masking with server limit—the OnCertain model and the DDPG algorithm

Table 10 Action prioritization based on server group—an example of inference rules

Inference rule
Server(?s_1), Server(?s_2), User(?u), Task(?t), Group(?g_1), Group(?g_2), isService(TaskOffloading), hasTask(?u, ?t), inGroup(?u, ?g_1), inGroup(?s_1, ?g_1), inGroup(?s_2, ?g_2), requestFrom(?u, ?s_1), requestFrom(?u, ?s_2) => hasPriorityOver(?s_1, ?s_2)



(a) The average total processed tasks satisfying security requirements.

(b) The average total processed tasks.

(c) The average total failed tasks.

Fig. 11 Action prioritization based on server group—the OnCertain model and the DDPG algorithm

Table 11 Execution prioritization based on usage history—an example of inference rules

Inference rule
Server($?s$), User($?u_1$), User($?u_2$), Task($?t_1$), Task($?t_2$), UsageHistory($?h_1$), UsageHistory($?h_2$), isService(TaskOffloading), hasTask($?u_1$, $?t_1$), hasTask($?u_2$, $?t_2$), hasUsageHistory($?u_1$, $?h_1$), hasUsageHistory($?u_2$, $?h_2$), requestFrom($?u_1$, $?s$), requestFrom($?u_2$, $?s$), isLessThan($?h_1$, $?h_2$) => hasPriorityOver($?u_1$, $?u_2$)

Table 12 Execution prioritization based on task priority—an example of inference rules

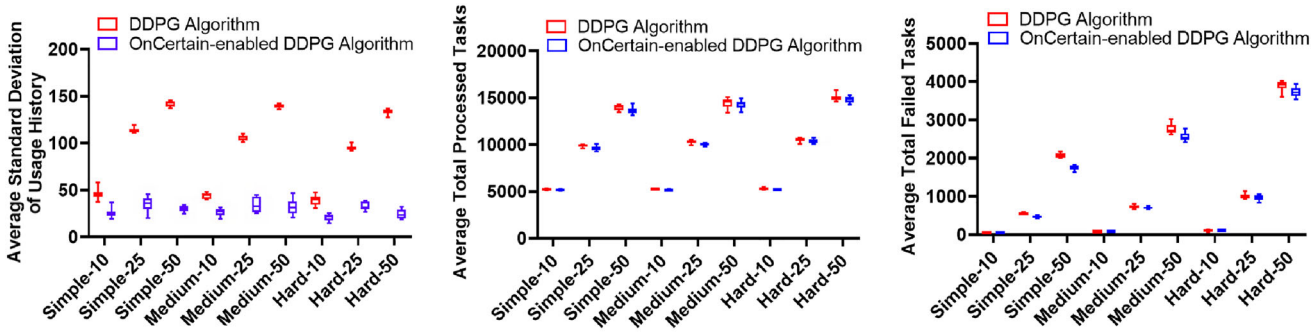
Inference rule
Server($?s$), User($?u_1$), User($?u_2$), Task($?t_1$), Task($?t_2$), Priority($?p_1$), Priority($?p_2$), isService(TaskOffloading), hasTask($?u_1$, $?t_1$), hasTask($?u_2$, $?t_2$), hasPriority($?t_1$, $?p_1$), hasPriority($?t_2$, $?p_2$), requestFrom($?u_1$, $?s$), requestFrom($?u_2$, $?s$), isGreaterThan($?p_1$, $?p_2$) => hasPriorityOver($?u_1$, $?u_2$)

Table 13 Execution prioritization based on application type—an example of inference rules

Inference rule
Server($?s$), User($?u_1$), User($?u_2$), Task($?t_1$), Task($?t_2$), isService(TaskOffloading), hasTask($?u_1$, $?t_1$), hasTask($?u_2$, $?t_2$), hasApplicationType($?t_1$, HealthCare), hasApplicationType($?t_2$, Entertainment), requestFrom($?u_1$, $?s$), requestFrom($?u_2$, $?s$) => hasPriorityOver($?u_1$, $?u_2$)

Table 14 Execution prioritization based on task latency—an example of inference rules

Inference rule
Server($?s$), User($?u_1$), User($?u_2$), Task($?t_1$), Task($?t_2$), Latency($?l_1$), Latency($?l_2$), isService(TaskOffloading), hasTask($?u_1$, $?t_1$), hasTask($?u_2$, $?t_2$), hasLatency($?t_1$, $?l_1$), hasLatency($?t_2$, $?l_2$), requestFrom($?u_1$, $?s$), requestFrom($?u_2$, $?s$), isLessThan($?l_1$, $?l_2$) => hasPriorityOver($?u_1$, $?u_2$)

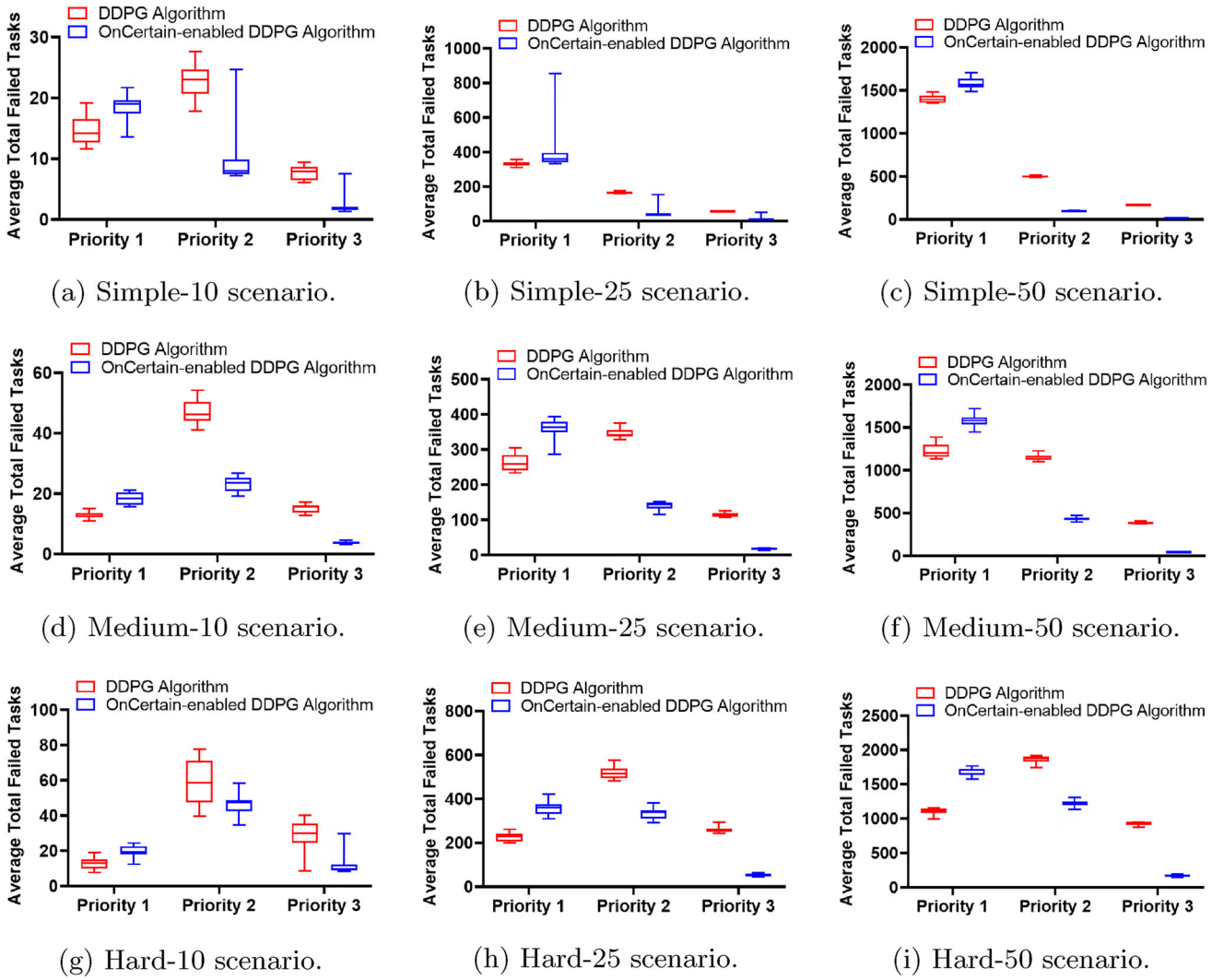


(a) The average standard deviation of usage history.

(b) The average total processed tasks.

(c) The average total failed tasks.

Fig. 12 Execution prioritization based on usage history—the OnCertain model and the DDPG algorithm



(a) Simple-10 scenario.

(b) Simple-25 scenario.

(c) Simple-50 scenario.

(d) Medium-10 scenario.

(e) Medium-25 scenario.

(f) Medium-50 scenario.

(g) Hard-10 scenario.

(h) Hard-25 scenario.

(i) Hard-50 scenario.

Fig. 13 Execution prioritization based on task priority—the OnCertain model and the DDPG algorithm

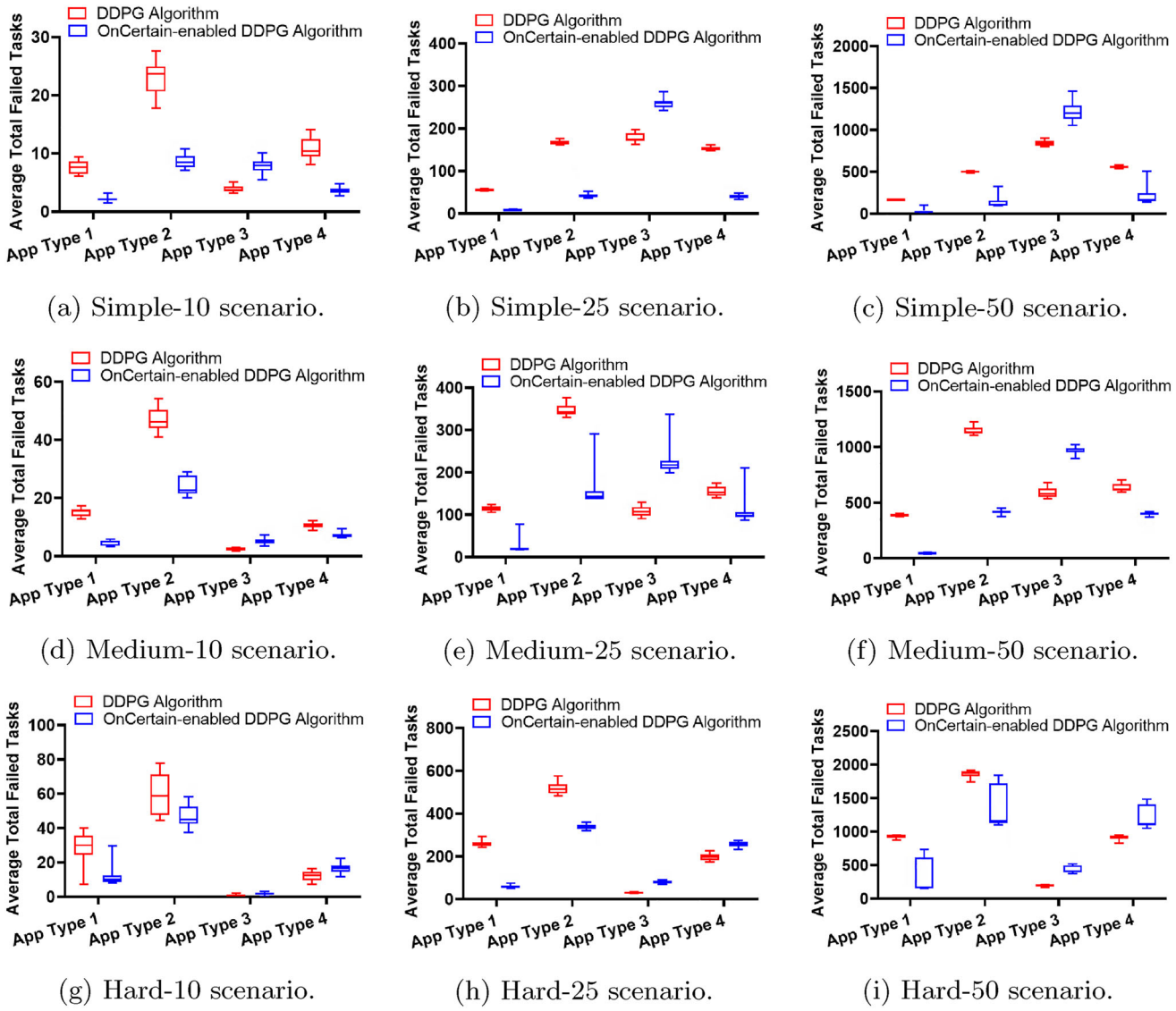


Fig. 14 Execution prioritization based on application type—the OnCertain model and the DDPG algorithm (App Type 1: remote health care, App Type 2: VoIP, App Type 3: data collection, App Type 4: entertainment)

baseline algorithm. The percentage decrease is 64%, 62%, and 64% in simple, medium, and hard scenarios, respectively. This improvement is 44%, 67%, and 79% for scenarios with 10, 25, and 50 users, respectively (see Fig. 12a). So, the proposed model shows more improvement when the number of users increases. Also, we can see that the average total processed tasks and average total failed tasks are not compromised to decrease the standard deviation of usage history (see Fig. 12b, c). This is particularly important as OnCertain can guarantee fairness among the users while keeping the whole system's performance.

According to the results shown in Fig. 13, prioritizing execution of the actions based on the task priority decreases the average total failed tasks by 25%. The percentage decrease for tasks with priority 2 is 69%, 58%, and 31% in simple, medium, and hard scenarios, respectively. This improvement is 43%, 56%, and 59% for scenarios with 10, 25, and 50 users. In the tasks with the highest priority (i.e., priority 3), the average total failed tasks decreases by 81%, 83%, and 73% in simple, medium, and hard scenarios and 67%, 82%, and 88% for scenarios with 10, 25, and 50 users, respectively. The OnCertain model decreases the average failure rate for the tasks with priority 3 more than those with priority 2. This is because when important tasks with the highest priority are generated in the EC environment, the execution prioritization step prioritizes them over other tasks to assign the requested server to them.

The results show that the execution prioritization based on the application type decreases the average total failed tasks overall by 31% (see Fig. 14). The percentage decrease for

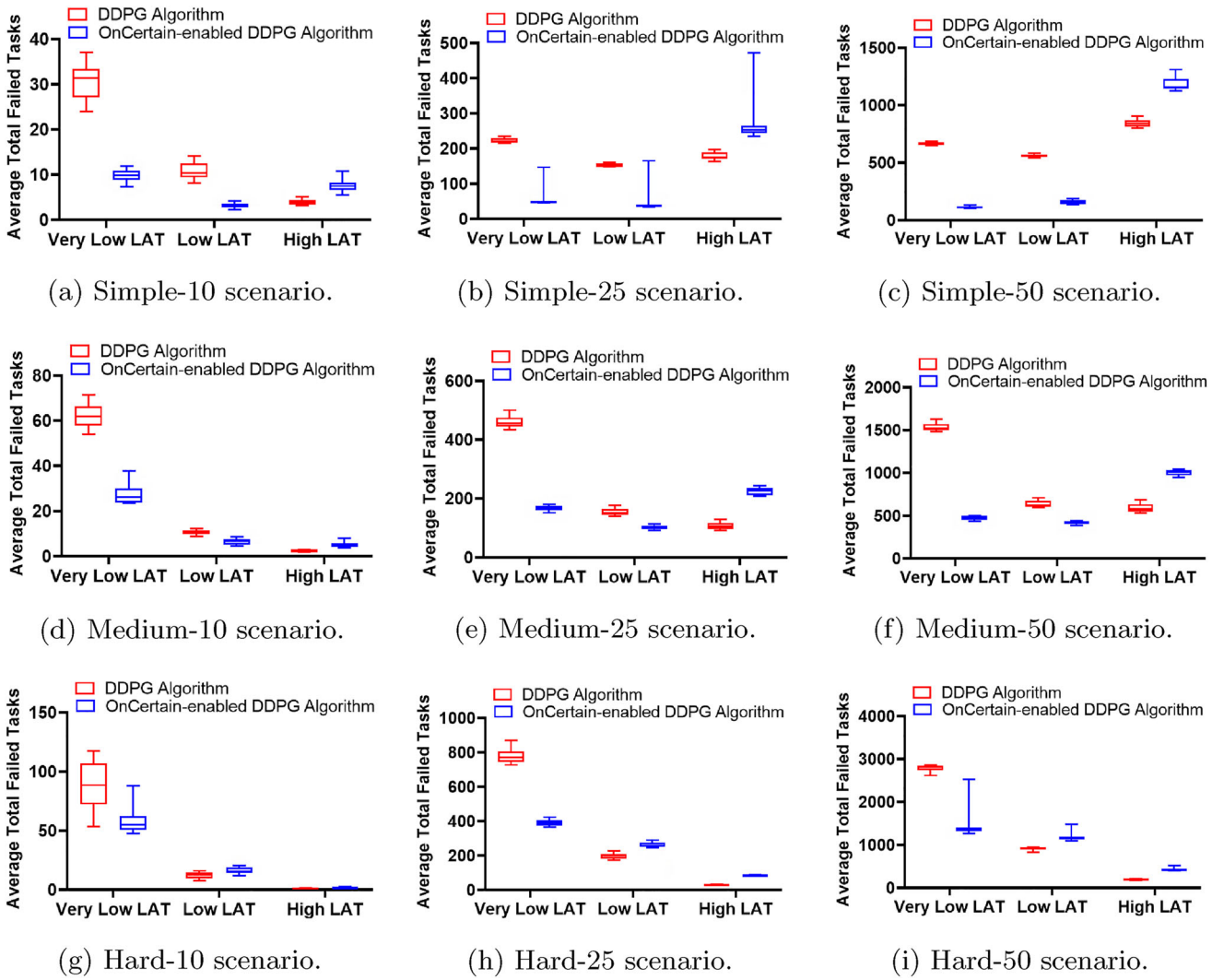


Fig. 15 Execution prioritization based on task latency—the OnCertain model and the DDPG algorithm

remote health care tasks (i.e., App Type 1) is 79%, 80%, and 67% in simple, medium, and hard scenarios, respectively. This improvement is 67%, 80%, and 79% for scenarios with 10, 25, and 50 users, respectively. In the VOIP tasks (i.e., App Type 2), the average total failed tasks decreases by 69%, 56%, and 28% in simple, medium, and hard scenarios and 44%, 54%, and 54% for scenarios with 10, 25, and 50 users, respectively. The OnCertain model decreases the average failure rate for the remote health care tasks more than the VOIP tasks. The reason is that the real-time requirement for remote health care applications is essential, and any delay is unacceptable. Hence, our model gives high priority to the remote health care applications in the task offloading service.

Figure 15 shows that the execution prioritization based on task latency decreases the average total failed tasks by 33%. The percentage decrease for very low latency tasks is 75%, 63%, and 44% in simple, medium, and hard scenarios and 53%, 63%, and 66% for scenarios with 10, 25, and 50 users, respectively. In the low latency tasks, the percentage decrease is 70%, 36%, and −33% in simple, medium, and hard scenarios and 25%, 23%, and 25% for scenarios with 10, 25, and 50 users, respectively. We observe that the OnCertain model decreases the average failure rate for the very low latency tasks more than others. Also, the improvement in the hard scenario is lower than in medium and simple scenarios. This is because the number of very low latency tasks increases, and our proposed model shows less improvement due to the limited number of servers.

7 Conclusion and future work

Self-adaptive systems face uncertainty from various sources, including uncertainties imposed by noisy sensor data and unanticipated events. Several approaches, including adaptive

modeling mechanisms, machine learning, and reinforcement learning algorithms, take uncertainty into account when making adaptation decisions. However, they cannot perform well in handling rare events, especially when the number of adaptation actions increases and on the fly decision-making is desired. To address these challenges, a novel Ontology-based unCertainty handling model (OnCertain) is proposed and evaluated in traffic signal control and edge computing environments. The OnCertain enables self-adaptive systems to use ontology and RL to augment their observation and reason about unanticipated situations.

OnCertain can be extended to handle uncertainties associated with different sources, including the system's goals and human behavior. It is essential that the ontology used by the OnCertain model is accurate and complete to maximize performance. The ontology must be continuously updated and evolved to operate in dynamic environments using ontology evolution techniques. Furthermore, a study about the impact of the goodness of ontology with respect to the performance of the systems could be useful. Also, the impact of under or overestimating concepts' weights on the overall process must be assessed. Moreover, the importance of concepts can change depending upon the context, so that the agent can have different ontologies for different "modes of operation", for example, during a hurricane or flood, or power outage. Also, Quality of Service (QoS) requirements can be defined based on the ontology to facilitate the actions and execution prioritization. To deal with unfamiliar environment changes, the proposed model can be extended to use ontology-based generalization techniques to evaluate the possible similarity of these unknown changes with known ones.

Availability of data and materials/Code availability The code of the OnCertain-DDPG algorithm and the simulated data and results are publicly available: https://github.com/saeedehghanadbashi/ontology-based-RL.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Calinescu R, Mirandola R, Perez-Palacin D, Weyns D (2020) Understanding uncertainty in self-adaptive systems. In: International conference on autonomic computing and self-organizing systems (ACSOS). IEEE, Washington, pp 242–251
2. Van Der Donckt J, Weyns D, Quin F, Van Der Donckt J, Michiels S (2020) Applying deep learning to reduce large adaptation spaces of self-adaptive systems with multiple types of goals. In: International symposium on software engineering for adaptive and self-managing systems (SEAMS). IEEE/ACM, Seoul, pp 20–30
3. Wang Y, Yang X, Liang H, Liu Y (2018) A review of the self-adaptive traffic signal control system based on future traffic environment. J Adv Transp 2018
4. Safavifar Z, Ghanadbashi S, Golpayegani F (2021) Adaptive workload orchestration in pure edge computing: a reinforcement-learning model. In: International conference on tools with artificial intelligence (ICTAI). IEEE, Washington, pp 856–860
5. Ciampi M, Coronato A, Naeem M, Silvestri S (2022) An intelligent environment for preventing medication errors in home treatment. Expert Syst Appl 193:116434
6. Naeem M, Coronato A (2022) An AI-empowered home-infrastructure to minimize medication errors. J Sens Actuator Netw 11(1):13
7. Schmerl B, Andersson J, Vogel T, Cohen MB, Rubira CM, Brun Y, Gorla A, Zambonelli F, Baresi L (2017) Challenges in composing and decomposing assurances for self-adaptive systems. In: De Lemos R, Garlan D, Ghezzi C, Giese H (eds) Software engineering for self-adaptive systems III. Assurances. Springer, Cham, pp 64–89
8. Esfahani N, Malek S (2013) Uncertainty in self-adaptive software systems. In: De Lemos R, Giese H, Müller HA, Shaw M (eds) Software engineering for self-adaptive systems. Springer, Berlin, pp 214–238
9. Mahdavi-Hezavehi S, Avgeriou P, Weyns D (2017) A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. In: Mistrik I, Ali N, Kazman R, Grundy J, Schmerl B (eds) Managing trade-offs in adaptable software architectures. Elsevier, Berlin, pp 45–77
10. Calinescu R, Grunske L, Kwiatkowska M, Mirandola R, Tamburrelli G (2010) Dynamic QoS management and optimization in service-based systems. Trans Softw Eng 37(3):387–409
11. Cámara J, Peng W, Garlan D, Schmerl B (2017) Reasoning about sensing uncertainty in decision-making for self-adaptation. In: International conference on software engineering and formal methods. Springer, Cham, pp 523–540
12. Moreno G.A, Cámara J, Garlan D, Klein M (2018) Uncertainty reduction in self-adaptive systems. In: International conference on software engineering for adaptive and self-managing systems (SEAMS). ACM, Gothenburg, pp 51–57
13. Gheibi O, Weyns D, Quin F (2021) Applying machine learning in self-adaptive systems: a systematic literature review. Trans Auton Adapt Syst 15(3):1–37
14. Metzger A, Quinton C, Mann Z.Á, Baresi L, Pohl K (2022) Realizing self-adaptive systems via online reinforcement learning and feature-model-guided exploration. Computing 104:1–22
15. Zouaq A, Nkambou R (2010) A survey of domain ontology engineering: methods and tools. In: Nkambou R, Bourdeau J, Mizoguchi R (eds) Advances in intelligent tutoring systems. Springer, Berlin, pp 103–119
16. Fong ACM, Hong G, Fong B (2019) Augmented intelligence with ontology of semantic objects. In: International conference on contemporary computing and informatics (IC3I). IEEE, Singapore, pp 1–4
17. Golpayegani F, Dusparic I, Clarke S (2019) Using social dependence to enable neighbourly behaviour in open multi-agent systems. Trans Intell Syst Technol 10(3):1–31
18. Alkhabbas F, Murturi I, Spalazzese R, Davidsson P, Dustdar S (2020) A goal-driven approach for deploying self-adaptive IoT systems. In: International conference on software architecture (ICSA). IEEE, Salvador, pp 146–156
19. Qureshi NA, Liaskos S, Perini A (2011) Reasoning about adaptive requirements for self-adaptive systems at runtime. In: International workshop on Requirements@Run.Time (RE@RunTime). IEEE, Trento, pp 16–22

20. Morandini M (2011) Goal-oriented development of self-adaptive systems. PhD thesis, University of Trento, Italy. http://eprints-phd.biblio.unitn.it/511
21. Whittle J, Sawyer P, Bencomo N, Cheng BH, Bruel J-M (2010) RELAX: a language to address uncertainty in self-adaptive systems requirement. Requir Eng 15(2):177–196
22. Ding Z, Zhou Y, Zhou M (2015) Modeling self-adaptive software systems with learning Petri nets. Trans Syst Man Cybern Syst 46(4):483–498
23. Qin Y, Xu C, Yu P, Lu J (2016) SIT: sampling-based interactive testing for self-adaptive apps. J Syst Softw 120:70–88
24. Sykes D, Corapi D, Magee J, Kramer J, Russo A, Inoue K (2013) Learning revised models for planning in adaptive systems. In: International conference on software engineering (ICSE). IEEE, San Francisco, pp 63–71
25. Vassev E, Hinchey M, Balasubramaniam D, Dobson S (2011) An ASSL approach to handling uncertainty in self-adaptive systems. In: Annual software engineering workshop (SEW). IEEE, Limerick, pp 11–18
26. Solano GF, Caldas RD, Rodrigues GN, Vogel T, Pelliccione P (2019) Taming uncertainty in the assurance process of self-adaptive systems: a goal-oriented approach. In: International symposium on software engineering for adaptive and self-managing systems (SEAMS). IEEE/ACM, Montreal, pp 89–99
27. Elkhodary A, Esfahani N, Malek S (2010) FUSION: a framework for engineering self-tuning self-adaptive software systems. In: International symposium on foundations of software engineering (FSE). ACM, Santa Fe, pp 7–16
28. Calinescu R, Kwiatkowska M (2009) Using quantitative analysis to implement autonomic IT systems. In: International conference on software engineering (ICSE). IEEE, Vancouver, pp 100–110
29. Cheng S-W, Garlan D (2007) Handling uncertainty in autonomic systems. In: International Workshop on Living with Uncertainties (IWLU'07), colocated with International Conference on Automated Software Engineering (ASE'07). Citeseer, Atlanta,
30. Baresi L, Pasquale L, Spoletini P (2010) Fuzzy goals for requirements-driven adaptation. In: International requirements engineering conference (RE). IEEE, Sydney, pp 125–134
31. Whittle J, Sawyer P, Bencomo N, Cheng BH, Bruel J-M (2009) Relax: incorporating uncertainty into the specification of self-adaptive systems. In: International requirements engineering conference (RE). IEEE, Atlanta, pp 79–88
32. Esfahani N, Kouroshfar E, Malek S (2011) Taming uncertainty in self-adaptive software. In: International symposium on foundations of software engineering (FSE). ACM, Szeged, pp 234–244
33. Han D, Xing J, Yang Q, Li J, Wang H (2016) Handling uncertainty in self-adaptive software using self-learning fuzzy neural network. In: Annual computer software and applications conference (COMPSAC), vol 2. IEEE, Atlanta, pp 540–545
34. Sharifloo AM, Metzger A, Quinton C, Baresi L, Pohl K (2016) Learning and evolution in dynamic software product lines. In: International symposium on software engineering for adaptive and self-managing systems (SEAMS). IEEE/ACM, Austin, pp 158–164
35. Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36(1):41–50
36. Weyns D, Schmerl B, Kishida M, Leva A, Litoiu M, Ozay N, Paterson C, Tei K (2021) Towards better adaptive systems by combining MAPE, control theory, and machine learning. In: International symposium on software engineering for adaptive and self-managing systems (SEAMS). IEEE/ACM, Madrid, pp 217–223
37. Rodrigues A, Caldas RD, Rodrigues GN, Vogel T, Pelliccione P (2018) A learning approach to enhance assurances for real-time self-adaptive systems. In: International symposium on software engineering for adaptive and self-managing systems (SEAMS). IEEE/ACM, Gothenburg, pp 206–216
38. Zavala E, Franch X, Marco J, Knauss A, Damian D (2018) SACRE: supporting contextual requirements' adaptation in modern self-adaptive systems in the presence of uncertainty at runtime. Expert Syst Appl 98:166–188
39. Knauss A, Damian D, Franch X, Rook A, Müller HA, Thomo A (2016) ACon: a learning-based approach to deal with uncertainty in contextual requirements at runtime. Inf Softw Technol 70:85–99
40. Mao X, Dong M, Liu L, Wang H (2014) An integrated approach to developing self-adaptive software. Inf Sci Eng 30(4):1071–1085
41. Wu T, Li Q, Wang L, He L, Li Y (2018) Using reinforcement learning to handle the runtime uncertainties in self-adaptive software. In: International conference on software technologies: applications and foundations (STAF). Springer, Berlin, pp 387–393
42. Teimourikia M, Fugini M (2017) Ontology development for runtime safety management methodology in smart work environments using ambient knowledge. Future Gener Comput Syst 68:428–441
43. Poggi F, Rossi D, Ciancarini P, Bompani L (2016) An application of semantic technologies to self adaptations. In: International forum on research and technologies for society and industry (RTSI). IEEE, Bologna, pp 1–6
44. Qureshi NA, Perini A, Ernst NA, Mylopoulos J (2010) Towards a continuous requirements engineering framework for self-adaptive systems. In: International workshop on Requirements@Run.Time (RE@RunTime). IEEE, Sydney, pp 9–16
45. Cheng W, Li Q, Wang L, He L (2018) Handling uncertainty online for self-adaptive systems. In: International conference on soft computing and machine intelligence (ISCMI). IEEE, Nairobi, pp 135–139
46. Hao J, Bouzouane A, Gaboury S (2019) An incremental learning method based on formal concept analysis for pattern recognition in nonstationary sensor-based smart environments. Pervasive Mob Comput 59:101045
47. Civitarese G, Bettini C, Sztyler T, Riboni D, Stuckenschmidt H (2019) newNECTAR: collaborative active learning for knowledge-based probabilistic activity recognition. Pervasive Mob Comput 56:88–105
48. Sanabria AR, Ye J (2020) Unsupervised domain adaptation for activity recognition across heterogeneous datasets. Pervasive Mob Comput 64:101147
49. Ghanadbashi S, Golpayegani F (2021) An ontology-based intelligent traffic signal control model. In: International intelligent transportation systems conference (ITSC). IEEE, Indianapolis, pp 2554–2561
50. Palm A, Metzger A, Pohl K (2020) Online reinforcement learning for self-adaptive information systems. In: International conference on advanced information systems engineering (CAiSE). Springer, Berlin, pp 169–184
51. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. MIT, Cambridge, pp 154–157
52. Thanh-Tung D, Flood B, Wilson C, Sheahan C, Bao-Lam D (2006) Ontology-MAS for modelling and robust controlling enterprises. In: International conference on theories and applications of computer science (ICTACS). World Scientific, Ho Chi Minh, pp 116–123
53. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M et al (2004) SWRL: a semantic web rule language combining OWL and RuleML. W3C Memb Submiss 21(79):1–31
54. Belohlavek R, Macko J (2011) Selecting important concepts using weights. In: International conference on formal concept analysis (ICFCA). Springer, Nicosia, pp 65–80
55. Saleena B, Srivatsa S (2015) Using concept similarity in cross ontology for adaptive e-Learning systems. J King Saud Univ Comput Inf Sci 27(1):1–12

56. Vitkute-Adzgauskiene D, Markievicz I, Krilavicius T, Tamosiunaite M. (2022) Learning and execution of action categories (ACAT). https://if.vdu.lt/en/research/projects/project-learning-and-execution-of-action-categories-acat. Accessed 2022
57. Li C, Tian G (2020) Transferring the semantic constraints in human manipulation behaviors to robots. Appl Intell 50:1711–1724
58. Vallejo M, Corne DW (2016) Evolutionary algorithms under noise and uncertainty: a location-allocation case study. In: Viktorovic M, Yang D, De Vries B (eds) Symposium series on computational intelligence (SSCI). IEEE, Athens, pp 1–10
59. Viktorović M, Yang D, de Vries B (2020) Connected Traffic Data Ontology (CTDO) for intelligent urban traffic systems focused on connected (semi) autonomous vehicles. Sensors 20(10):2961
60. Wei H, Zheng G, Gayah V, Li Z (2019) A survey on traffic signal control methods. Comput Res Repos. arXiv:1904.08117
61. Mazak A, Schandl B, Lanzenberger M (2010) iweightings: Enhancing structure-based ontology alignment by enriching models with importance weighting. In: International conference on complex, intelligent and software intensive systems (CISIS). IEEE, Krakow, pp 992–997
62. Wang J, Liu Y, Li B (2020) Reinforcement learning with perturbed rewards. In: AAAI conference on artificial intelligence. AAAI Press, New York, vol 34, pp 6202–6209
63. Kiran BR, Sobh I, Talpaert V, Mannion P, Al Sallab AA, Yogamani S, Pérez P (2022) Deep reinforcement learning for autonomous driving: a survey. Trans Intell Transp Syst 23(6):4909–4926
64. Alegre LN (2019) SUMO-RL. https://github.com/LucasAlegre/sumo-rl. Accessed 2022
65. Han M, Lee Y, Moon SB, Jang K, Lee D (2008) CRAWDAD dataset kaist/wibro (v. 2008-06-04). https://crawdad.org/kaist/wibro/20080604. https://doi.org/10.15783/C72S3B
66. Chen D (2020) Resources allocation in the edge computing environment using reinforcement learning. https://github.com/davidtw0320/Resources-Allocation-in-The-Edge-Computing-Environment-Using-Reinforcement-Learning. Accessed 2022