SMT-as-a-Service for Fog-Supported Cyber-Physical Systems

Stefan Holzer
Distributed Systems Group
TU Wien
Austria

Pantelis A. Frangoudis*
Distributed Systems Group
TU Wien
Austria

Christos Tsigkanos
Department of Aerospace Science & Technology
University of Athens
Greece

Schahram Dustdar
Distributed Systems Group
TU Wien
Austria

ABSTRACT

Various properties related with the safe, correct, and efficient operation of Cyber-Physical Systems (CPS) can be expressed via formal languages and checked at runtime or offline by appropriate verification tools. Such tools operate on monitoring data about the CPS state and functionality, typically collected from IoT devices. A specific approach involves modeling CPS state or operations using Satisfiability Modulo Theories (SMT) formalisms, and using solver software to check whether given CPS properties are satisfied or to derive satisfiable CPS configurations. The computational requirements of this process can however be significant, which challenges its timely execution on IoT/edge devices where input date originate. To address this challenge, we present an architecture that allows the distributed execution of SMT problem solving workloads over the computing continuum as a service. Our design supports arbitrary hierarchies of solver nodes running anywhere from the IoT device to the cloud, each independently executing decision-making logic as to whether to solve an SMT problem instance locally or to recursively offload the task to other nodes in the continuum. We demonstrate the benefits of offloading by implementing and quantitatively evaluating different reinforcement learning-based decision-making strategies addressing latency minimization and energy efficiency goals, and showcase the practicability of our scheme in a fog robotics proof-of-concept.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; **Robotics**; • **Mathematics of computing** → **Solvers**; • **Computing methodologies** → **Reinforcement learning**.

KEYWORDS

Computing continuum, IoT, Computation Offloading, Satisfiability Modulo Theories, Fog Robotics

*Corresponding author. Email: pantelis.frangoudis@dsg.tuwien.ac.at

ICDCN '24, January 04–07, 2024, Chennai, India

ACM ISBN 979-8-4007-1673-7/24/01…$15.00
https://doi.org/10.1145/3631461.3631562

ACM Reference Format:
Stefan Holzer, Pantelis A. Frangoudis, Christos Tsigkanos, and Schahram Dustdar. 2024. SMT-as-a-Service for Fog-Supported Cyber-Physical Systems. In *25th International Conference on Distributed Computing and Networking (ICDCN '24), January 04–07, 2024, Chennai, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3631461.3631562

1 INTRODUCTION

Complex Cyber-Physical Systems (CPS), such as those emerging in various next-generation IoT scenarios, often need to operate according to strict functional and non-functional specifications, such as safety-related [23]. Ensuring that these specifications are met can take place by following formalisms that prescribe application behavior and collecting monitoring information encoded in traces, which are verified at run time or offline to detect specification violations or to derive correct configurations.

Such problems can often be modeled with Satisfiability Modulo Theories (SMT) [11], an extension of the satisfiability problem (SAT). SMT is a powerful approach for solving complex constraint satisfaction problems. These arise, for example, in motion planning for robots [18], verifying the correct operation of satellite [28] or edge systems [37] by trace checking, or detecting threats in rule-based smart home systems [38]. Application areas of SMT also extend beyond IoT/CPS and range from scheduling and optimization to computer security and software quality.

The procedure is, however, costly. Solving hard satisfiability problems that contemporary complex CPS face may require significant resources and time. The latter is important to minimize when it comes to systems that carry out time-sensitive operations. An approach to dealing with such high resource requirements is to transport relevant input data to cloud instances and solve the problems there, taking advantage of the abundance of cloud resources. However, this faces two significant issues: (i) for various problem instances, the time it takes to transport input data to the cloud and receive a response offsets the potential latency gains that come with using more compute resources, and (ii) various CPS applications, such as those relevant with disaster management scenarios, often need to operate under limited, intermittent, or no connectivity beyond the edge network space, which precludes cloud offloading.

Fortunately, advances of the last decade in IoT/edge device platforms, virtualization technology, and orchestration middleware, have given rise to a computing continuum that extends from the IoT device all the way to the cloud, making it possible to execute computation tasks almost uniformly (performance aside) anywhere

in the continuum. Taking advantage of these developments, we address the problem of efficient handling of SMT workloads originating in the IoT device space, aiming to answer the following research questions:

RQ1: How to architecturally support SMT workloads in the device-to-cloud computing continuum.

RQ2: How to provide offloading decision support for the evaluation of SMT formulas considering different performance criteria.

RQ3: How can different offloading decision-making strategies improve the performance of SMT formula evaluation within fog computing settings.

Our answers to these questions come with the following contributions: ① We introduce a solver node abstraction, based on which we design and implement a recursive, extensible architecture that handles local or offloaded execution of SMT formulas in an "as-a-Service" manner over the computing continuum (§3). Our design supports a wide range of SMT solvers and offloading strategies, depending on the particularities of the underlying compute hosts, which may range from powerful edge servers to single-board computers. ② We design two different offloading decision-making strategies based on Reinforcement Learning (RL). These mechanisms have different capabilities and computational requirements, thus being tailored to different host environments (resource constrained IoT devices vs. more powerful fog nodes). Furthermore, they can support diverse goals such as response time minimization and energy efficiency (§4). ③ With a view to fog robotics as a representative CPS use case, we perform extensive experiments over a device-to-cloud testbed. Our results demonstrate the practicability of our approach to support diverse SMT workloads, as well as the latency and energy cost savings that come with intelligent offloading (§5).

2 RELATED WORK

We advocate an architecture and software framework to support the execution of SMT workloads over compute infrastructure spanning the device-to-cloud continuum; consequently, we classify related work into three major categories. First, we discuss SMT in CPS and IoT as relevant domains. Subsequently, we discuss cloud and fog robotics, positioning our work within a major application area. Finally, we consider computation offloading , since it is a key component of our approach.

SMT in CPS/IoT. The application of SMT is in most cases some type of verification task. In the area of IoT security, Mohsin et al. [30] present IoTSAT, a formal modeling approach to create a security analysis framework, where SMT formulas representing IoT specific threat models are evaluated. Wang et al. [38] study the interactions between trigger-action rules and use SMT to discover inter-rule vulnerabilities, which are highly relevant in platforms such as smart homes. Liang et al. [25] present a solution for systematically debugging IoT control system correctness for building automation, where IFTTT (If This Then That) rules and policies are specified as conjunctions of conditions and the proposed framework transforms them into SMT formulas.

In the area of robotics, SMT and SAT find use in different kinds of problems such as scheduling, resource management, and motion planning. Hung et al. focus on motion planning with rectangular obstacles using SMT solvers to find a feasible path from the source to a goal [17]. Imeson and Smith [18] propose a new method for solving multi-robot motion planning problems with complex constraints. In particular, they encode the task assignment and the path planning problems using SAT, handling additional complex constraints like battery life limitations, robot carrying capacities or robot-task incompatibilities. Another emerging area is smart factories as the new industrial paradigm. Bit-Monnot et al. [4] present two SMT-based planners for smart factories. A recurring problem is task planning, and such planning problems can be modeled with SMT. In all these cases, our approach can help solve these problems more efficiently.

Cloud and fog robotics. Cloud robotics aim at using elastic resources offered by a cloud infrastructure to overcome the resource constraints of robots. By using the cloud and offloading computationally intensive tasks, many new applications for robotics emerge. Hu et al. [15] propose a machine-to-machine (M2M) communication framework, where the M2M layer is used for communication between machines to form a collaborative computing structure and the machine-to-cloud (M2C) layer utilizes a pool of shared computation and storage resources provided by the cloud. Fog robotics [5] extend these principles towards utilizing compute resources along the device-to-cloud continuum, even including robots as mobile computation platforms, when robot resource availability allows it. A popular case for fog robotics is Simultaneous Localization And Mapping (SLAM) [1, 16]. Chen et al. [8] experiment with an industrial robotics system based on edge computing with the deployment of edge nodes near the data sources, proposing a three-tier architecture where the (mid) edge tier hosts latency-sensitive robot control functions, and pre-processes data from robots to reduce the volume of data transmitted to the cloud.

Computation offloading. The literature on computation offloading in fog and edge computing is extensive [22]. Task offloading decisions can be driven by specific goals such as energy efficiency or latency reduction. As an example, Zhang et al. [41] provide a solution for energy-efficient offloading for mobile edge computing in 5G heterogeneous networks. They formulate an optimization problem to minimise the energy consumption of the offloading system using a detailed energy model, considering the computational capabilities of the mobile device, the associated costs of the task computation, and the file transmission depending on multiple factors such wireless channel state and interference, size of the input and output of computation, and the mobile device characteristics. Shahhosseini et al. [33] target task offloading for IoT applications and focus on finding a solution for the optimal response time taking into account their particularities, such as the ratio of input/output data size and data flow configuration in a three-layer architecture consisting of a sensor layer, fog layer and cloud layer. Masoudi and Cevdar [27] also address the question of on-device vs. edge computation for mobile services and propose a solution for delay-aware decision making to minimize power consumption. For cases where input data, and thus the respective data processing tasks, are divisible, Tran-Dang and Kim [35] propose FRATO, a framework for delay-minimizing task

offloading in three-tier fog computing architectures using particle-swarm optimization. Often, task offloading is jointly considered with resource allocation and scheduling [6, 13, 42], giving rise to challenging optimization problems. These challenges are further exacerbated by edge heterogeneity [26]. The economics of fog computing are of growing interest [39] and of particular importance for computation offloading in an environment with selfish edge/fog nodes. In such settings, Diamanti et al. [12] design an incentive mechanism for offloading latency-tolerant tasks and apply a Stackelberg game-theoretic approach to joint compute-communication resource allocation.

An approach to deal with complex offloading optimization problems is via Reinforcement Learning [19, 40]. Li et al. [24] decide between executing tasks locally on user equipment or offloading them to a single Multi-access Edge Computing (MEC) server using a RL-based optimization framework based on a Deep Q-Network (DQN), aiming to reduce delay and energy consumption. Khoramnejad and Erol-Kantarci [21] support binary and partial task offloading towards a MEC server via a distributed multi-agent Deep Reinforcement Learning (DRL) scheme for joint resource allocation and offloading management. Chen et al. [7] expand towards the use of multiple MEC servers in a sliced Radio Access Network (RAN) where offloading decisions are made by a centralized controller. Karagiannis et al. [20] use Q-Learning to efficiently route IoT data processing requests along fog compute node hierarchies, in order to avoid long forwarding chains and reduce latency. In the context of the Internet of Vehicles (IoV), Ning et al. [31, 32] present a DRL-based offloading framework where tasks can be executed at cloudlets, Roadside Units (RSU) and vehicles as fog compute nodes. Finally, Tripathi et al. [36] identify problems related with resource contention between network functions and user application workloads executed at the edge, and the correlations therein, and propose a distributed RL-based framework for joint network-compute resource orchestration and dynamically adapting service-specific configuration parameters.

Summary. Compared with existing works, we aim to provide system support for the dynamic execution of SMT workloads in a seamless way across the device-to-cloud continuum in a *Solver-as-a-Service* manner, a problem previously not addressed in the literature. Our focus is to provide a runtime environment to facilitate this. From an offloading mechanism perspective, our contributions are complementary, as we focus on providing the facilities to allow for configurable criteria and algorithms for offloading decisions – delay awareness and energy efficiency are two potential goals. Furthermore, our scheme is tailored to the particularities of handling SMT workloads, which is reflected in the state representation we use when applying RL.

3 ARCHITECTURE

In this section, we start by outlining key principles as requirements permeating our approach, before describing in detail the node element around which our architecture is designed.

3.1 Key design principles and requirements

We design an architecture and software framework to support the execution of SMT workloads originating at IoT devices over a distributed compute infrastructure spanning the device-to-cloud continuum, in a stateless, as-a-service manner. Our approach is *recursive* in nature: A service invocation starts on-device – or at the nearest edge node capable of handling it, and the request propagates towards the cloud until a fog node decides to serve it by locally invoking an SMT solver process. Different fog nodes make decisions independently based on diverse and node-specific decision-making logic that may be put in place. Our approach adheres to the following design requirements:

R1. Support for resource-constrained devices: Our system should be deployable on different types of devices along the computing continuum. Such devices could be significantly constrained, as is the case for the mobile robots and single-board computers we experiment with. This drives the design and implementation towards lightweight solutions.

R2. Support for different offload decision-making mechanisms: At the core of our work lies the decision on whether SMT formulas should be offloaded or solved locally. The algorithms to decide that can be based on different methods and their selection could be driven by the available compute resources. Therefore, our system should support multiple decision modules, as extensibility and interchangeability are desirable features. However, common parts of the software should be extracted, and it should be possible to add another specific approach with minimal integration effort. It is also essential that several decision modules can co-exist, without affecting each other's performance and functionality negatively.

R3. Independence from SMT solver implementations: The actual SMT solver instance used in an implementation of our scheme should be abstracted, which enhances extensibility and allows to tailor the used solver to the particularities of the underlying host. To achieve this, a common language to express problems and interface with solvers is required. The de facto approach to this end is to use SMT-LIB [2], which is also the case in our system.

3.2 Node Design

Figure 1 illustrates the different modules and the interactions between them in the architecture we advocate. Service invocation involves the following workflow.

(1) The first step is always the occurrence of a new SMT formula, which may take place at regular intervals or in response to an event, depending on the use-case scenario. The SMT formula can be provided from an external system or application, so the possibilities here are diverse.
(2) The next activity is to get the latest state information, which is relevant for decision-making. This results in a call to a monitoring module.
(3) After determining the state, the decision module is invoked to select between solving locally or offloading to an appropriate node.
(4) If the decision is to offload, the SMT formula is forwarded to the communication module, which offloads it to the selected node.
(5) If the decision is to solve locally, the formula is handed to the SMT solver module that solves the problem.
(6) The result is recursively returned to the source instance.

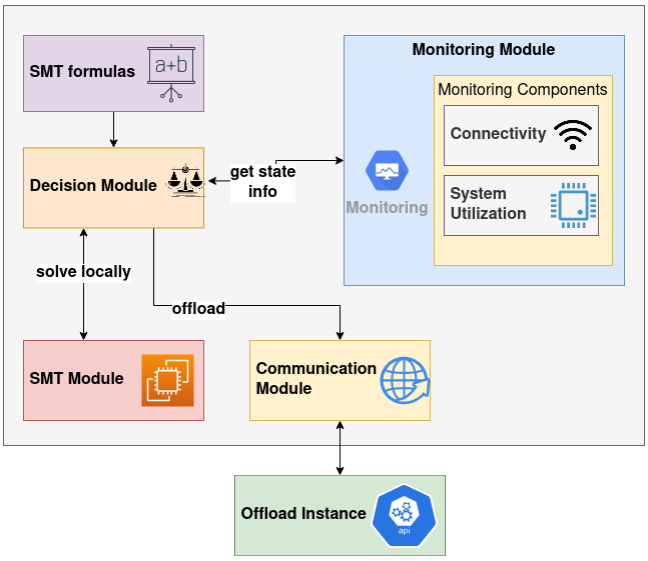


Figure 1: Birds-eye view of the node architecture supporting SMT-as-a-Service for fog offloading.

The core components of our node architecture are presented in detail next.

3.2.1 Communication Module. The communication module is used for communication with other devices. If an SMT formula is to be offloaded, this is carried out via the communication module. Our implementation supports REST-based interactions, but because of the loose coupling between node components, the communication mode can be replaced in a relatively straightforward way.

3.2.2 Monitoring Module. The monitoring module is used to monitor the environment (the devices on which the system is running). This component could provide any information useful for decision-making. This could range from battery level or CPU usage to connectivity to different hosts in the form of average round-trip times (RTT). The monitoring component is asynchronously executed and gets the state information periodically. The benefit of doing this in the background and not at each decision is that the cost of the decision itself can be reduced, as the operations for monitoring are more time-intensive. If the device is more powerful, the period could be adapted accordingly. This would lead to more up-to-date monitor values for the decision module and it is a manifestation of the trade-off between accuracy/decision quality and monitoring overhead; mechanisms for navigating this trade-off space are left for future work.

3.2.3 SMT Module. The SMT module consists of two parts. The first one is an interface to the native SMT solver. The second one provides an API endpoint over which it can receive requests to solve SMT problems. These requests might either originate locally at the node (e.g., an IoT device hosting it) or come from other nodes (offloading requests).

3.2.4 Decision Module. The decision module receives SMT formulas and decides how to proceed with them. Depending on which decision mode is active, the formulas are forwarded to the corresponding submodule. In general, external modules can also be called, to take over further processing. Our system readily supports a number of such modules (see Section 4) that are based on reinforcement learning. Our rationale is that the decision module should be operable in different and changing system environments, which rules out threshold-based, heuristic approaches. Furthermore, it makes the whole system more flexible and applicable to different use cases, as it can be configured to focus on the needs of the supported application or node operator preferences. The choice of which such module to activate is related with the capabilities of the underlying host. For example, as we discuss, demanding DQN-based mechanisms are not suitable for execution on resource-constrained robots.

4 DECISION-MAKING STRATEGIES

Our approach provides architectural support for diverse offload decision-making schemes. As an example, in this paper, we design two such mechanisms based on reinforcement learning [34]. RL involves agents that interact with their environment and receive feedback for their actions in the form of reward or penalty. Based on this feedback, their aim is to learn a policy in terms of selecting actions given the perceived state of the environment, so that they maximize their expected cumulative reward.

4.1 System model

In our architecture, each fog node that executes our offloading logic hosts such an agent, which acts independently and selects to either solve an SMT problem instance locally or offload it to another node from a configured set (e.g., a nearby fog node or a node in the cloud). In what follows, we elaborate on what constitutes a node's view of system state, the available actions to pick from, and the reward signal. Then we delve deeper into the details of the two example strategies we have applied, namely Q-Learning and Deep Q-Learning.

State space. Environment state information is composed of the problem complexity of the SMT formula to be solved and the connectivity status of the path towards each potential offloading target. We assume that further information on the state of offloading targets, such as their current CPU load or battery level, is not available to the agent, though our design does not preclude this option. The connectivity status can be estimated via round-trip time measurements, and can be represented in different fidelities, based on the selected decision-making algorithm. For example, RTT values can be mapped to distinct connectivity classes (e.g., poor, fair, etc.) or can be used directly.

Inferring the complexity of an SMT formula from its structure (e.g., number of variables, quantifiers, etc.) is not straightforward. In our case, we assume the existence of an oracle that, given an SMT formula, decides on its complexity. There are two practical ways we have considered to build such an oracle: (i) Since for a given application the SMT formulas to be solved are often specific to it, a benchmarking step can precede deployment, where the formulas of interest are handed to a solver running on representative hardware to measure their execution time. Then, a hardness rating H_f is assigned to formula f by the following expression:

$H_f = \min\{1, \frac{t_f}{T_{max}}\}$, where t_f is the average time to solve the formula on the given platform, and T_{max} is a configurable threshold that corresponds to the maximum tolerable execution time of a hard formula. (ii) At runtime, a node can keep track of the history of executions of the same formula and classify it accordingly; this implies that offloading performance may suffer initially, but saves on benchmarking effort.

Action space. The available actions of a node are defined by the number of potential offloading targets. It is up to the node operator to define this set. We should note that in our approach, an offloading target need not necessarily correspond to a single host, but also to node clusters thereof. For example, it is reasonable to assume a three-tier compute infrastructure, where the first tier is the IoT device where requests originate, the second tier is an edge cluster accessible via a gateway node, and the third tier is a VM cluster hosted in a public cloud. Each node may then need to know a single entry point to the tier that follows, thus abstracting the internal configuration of the latter. This can significantly reduce the state space and monitoring load.

Reward model. The reward function is related with the criteria the system operator aims to optimize for, such as latency, energy consumption, or their combination. In the case of latency, the reward can be defined as a configurable threshold time minus the time required to solve the problem – if solution time is higher than the threshold, the agent receives a negative reward. For a decision-making mechanism that aims to reduce energy costs, similar to a latency-based reward scheme, the agent observes the energy cost of a decision and calculates a reward based on that. Energy minimization is important, e.g., for mobile robots that operate on battery, but accurately measuring energy costs is often not straightforward. We provide some details on a simplified energy cost model we have applied in our proof-of-concept implementation in Section 5.

Exploration vs. exploitation. Our RL-based mechanisms need to make offloading decisions while learning. We have selected to alternate between *exploration* and *exploitation* steps using the widely applied ϵ-greedy strategy, where the agent chooses a random action (exploration) with probability ϵ, while with probability $1 - \epsilon$ it follows the optimal action based on current knowledge (exploitation). From an architecture and implementation perspective, all the relevant parameters (exploration rate, decay rate and minimal ϵ value) are configurable, while it is possible to extend the system to experiment with other exploration strategies without much integration effort.

4.2 Q-Learning

We first design a decision-making module based on Q-learning. A Q-learning algorithm aims to derive a value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, which represents the expected cumulative discounted future reward if the agent selects action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$ and continues by following the optimal policy. Q can be represented as a table, which is updated each time an agent takes an action considering the observed reward, the selected action, and the state the agent transitions to. Since in our case the state encodes the connectivity status towards potential offloading targets and the hardness of the SMT problem to solve, all of which take continuous values, we discretize their values to reduce the state space. To further limit space requirements, instead of considering the connectivity from a fog node to each potential offloading target, we use a single value to characterize the current connectivity capabilities of the node. Eventually, both problem hardness and node connectivity state belong to one of five distinct categories: poor, fair, average, good, excellent. This way, the space to maintain is reduced to 25 states.

Admittedly, this results in significant loss of representation accuracy and can come at the cost of offloading decision quality, but allows to execute the Q-learning algorithm at very low-end hardware, as is the case for robot nodes in our testbed (Section 5.2). Our architecture allows to seamlessly integrate more sophisticated and resource-demanding algorithms.

4.3 Deep Q-Learning

The second decision-making scheme we develop is based on Deep Q-Learning. Here, instead of the Q-table, an artificial neural network is used to approximate the value function, and the Deep Q-Network (DQN) approach is used to train it [29]. DQN deals with space limitations of Q-learning when it comes to state representation, but at the same time targets fog nodes with increased compute capabilities to train and use the neural network. This approach allows to use a more precise state representation. Here, state encodes the original hardness ranking for SMT formulas, as well as the precise connectivity information to all potential offloading targets. The selected representation defines part of the neural architecture: The number of input neurons must be equal to the number of state dimensions (i.e., formula hardness, latency information about all potential execution targets) and the number of output neurons must be equal to the number of actions. In our implementation, we have opted for a fully-connected neural network with three hidden layers, with 24 neurons each.

5 EVALUATION

5.1 Methodology and objectives

We showcase our SMT-as-a-Service scheme in a fog robotics context. For this purpose, we implement our service-based design and deploy it on top of an end-to-end testbed representative of the cloud continuum. Our testbed features a mix of low-end robots (IoT device tier), a single-board computer cluster with Raspberry Pis (edge tier), typical in evaluating edge computing systems, and server-grade hosts in an institutional data center and in the cloud. This serves as a basis for both a feasibility demonstration and a quantitative evaluation of different candidate offloading decision-making strategies. We further show our system in action in a robot path planning use case.

5.2 Architecture implementation and testbed

Our architecture naturally lends itself to a microservices-based implementation, where each individual component is implemented as a containerized service exposing REST API endpoints. All our components are written in Python and our implementation is available as open source.[1] The DQN-based decision module is built on PyTorch. The SMT module receives formulas encoded in SMT-LIB

[1] https://github.com/Stefan2911/SMTaaS

format in the payload of a REST API call, and interfaces with the solver software using the PySMT library [14]. This has the advantage of selecting the appropriate solver backend based on the hardware capabilities of the node. For example, we use CVC4 [3] on IoT/edge hosts, while we also support MathSAT5 [9] and Z3 [10] on cloud hosts.

Figure 2 presents our testbed. We use a Lego Mindstorms EV3 robot, where SMT solution requests originate. It runs Linux (ev3dev)[2] on a TI Sitara AM1808 (ARM926EJ-S core) processor at 300 MHz, with 16 MB of RAM and 64 MB of flash memory, and is representative of a resource-constrained mobile IoT device. The robot has a USB Wi-Fi interface, and can offload workload to an edge cluster made up of two Raspberry Pi 4 Model B Rev 1.1 devices (from now on termed Dedicated Edge Devices – DEDs). These run Raspbian Linux on a 4-core ARMv7 Processor at 1.5 GHz with a 4 GB RAM. Our testbed further includes two cloud sites: the first hosts 3 VMs with 2, 4, and 8 CPU cores, respectively, and the second hosts a single 4-core VM.

Each of the above, from the robot to the cloud VMs, runs an instance of our node, and they are configured in a way that the robot can offload tasks to the edge nodes, while the latter can offload tasks to the cloud nodes.

5.3 Offloading scenarios

Figure 2 shows our main decision mechanism configuration: the robot makes offloading decisions using the simple Q-learning mechanism, owing to its resource limitations, while edge devices run the DQN-based one. We evaluate this setup (abbreviated as Q+DQN+C) against the following benchmarks:

Robot Only (RO): All the SMT formulas are solved on the robot itself. The robot does not make decisions and there is no offloading to other devices. The robot always forwards the formulas to the local SMT module that interfaces with the CVC4 solver. These scenarios serve to demonstrate the limitations of executing only on-device computations and motivate the need for offloading. It should be noted that this setup can be considered characteristic of scenarios that feature disconnected and fully autonomous robot operation, where physical limitations such as the lack of network infrastructure (e.g., in disaster management situations) mandate that service logic is executed on board, without edge or cloud assistance and control.

Edge devices only (EO): Each SMT formula is offloaded to one of the devices of the edge cluster, picked uniformly at random by the robot. Similarly, the SMT module uses CVC4 as the solver.

Cloud Only (CO): Each SMT formula is directly offloaded to one of the cloud nodes, also picked uniformly at random per request by the robot.

Q-Learning on the robot; edge cluster only (Q+E): We use Q-Learning on the robot to decide whether the problem should be offloaded to one of the DEDs or solved on the robot. There is no option to further offload workload from DEDs to the cloud.

Q-Learning on the robot; Q-Learning on edge devices (Q+Q+C): We use Q-Learning on the robot to decide whether the problem should be offloaded to one of the DEDs or solved on the robot. Q-Learning is also used on the DEDs to decide whether the problems should be offloaded to one of the cloud instances or solved on the DEDs.

The benchmarks that involve decision making represent use cases where both offloading and solving locally are potential options. However, if one option is always the preferred solution, our experiments show that the algorithms we advocate will learn it. To study the effects of latency on offloading behavior, for each scenario that involves computation beyond the robot, we carry out five different sets of experiments, each corresponding to a different end-to-end delay that we simulate at the application layer.

5.4 SMT workload

To simulate SMT problem workloads, we selected a subset of problems from the official SMT-LIB-benchmark repository used for competitions,[3] applying the following process: We measured the execution times of all the problem formulas with a timeout of 1.5 s using the CVC4 solver on a host with an Intel(R) Core(TM) i5-6200 CPU at 2.30GHz and 8 GB RAM, thus filtering out all formulas that took more than 1.5 s to run. We further filtered out formulas that cannot be solved on the robot due to processor or solver implementation limitations. Then, we sorted the problems by execution time and split them in three sets: easy (0-0.5 s; 14 formulas), medium (0.5 - 1 s; 7 formulas), and hard (1 - 1.5 s; 9 formulas). Finally, we created a fourth set that includes formulas from all three categories (mixed; 10 formulas).

5.5 Latency savings

We run a set of testbed experiments measuring the response time for handling an SMT problem solution request, from the moment it is generated at the robot end, until the result of the computation has been received. Figure 3 presents the results of our experiments for the different SMT problem datasets used, and simulating increasing response times by adding a fixed latency component each time. It should be noted that, on average, solving the problems on-device results in approximately two orders of magnitude higher latency, which is not affected by the delay we introduce in the path. For these reasons, we omit these results from the plots. However, for specific simple problem instances in our dataset, the decision-making mechanism on the robot still opts for executing them locally.

As the figure illustrates, the cloud only (CO) and DED only (EO) approaches always show a linear increase in response time with additional latency. Furthermore, there is not much difference between using DQN (Q+DQN+C curve) and Q-learning (Q+Q+C curve) on the DEDs. This has to do with the way we simulate latency, which for simplicity takes place only on the link between the robot and the edge cluster.

If the workload is composed only of simple problems, from about 100 ms additional latency on, it becomes beneficial to solve some problems on the robot. All of our approaches learn this behavior and make an optimized decision. In this case, the CO approach is slightly worse than the EO for the following reasons: (i) the problems are very simple and can be solved very fast on the DEDs; (ii) the natural latency between the robot and the cloud is worse than the natural

[2] https://www.ev3dev.org/

[3] https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks

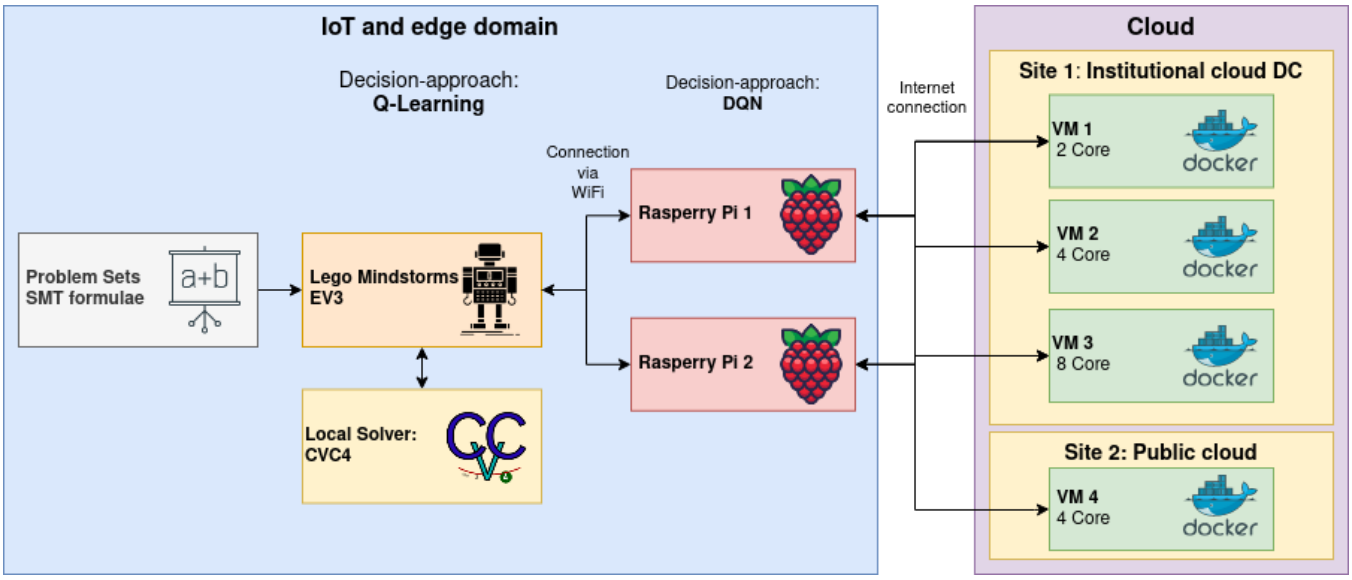


Figure 2: Testbed setup for our target scenario. SMT problem solving requests originate at a robot, where a Q-learning agent decides to offload them to dedicated edge devices (RPi) or solve them locally. Edge devices, in turn, decide using DQN whether to offload the received requests to cloud instances or handle them locally.

latency between the robot and the DEDs. In general, Q-Learning and DQN on the DEDs lead to additional overhead that can not be compensated for problems from the simple problem set. This causes slightly worse results when there is an additional decision-making approach on the DEDs.

For problems of medium difficulty, offloading from the robot is the best option. CO fares better than EO, since problems can be solved faster in the cloud and compute time dominates network latencies. For this reason, in contrast with the simple problem set, the additional offloading capabilities from the DEDs pay off: whatever the algorithm, both approaches (Q+Q+C, Q+DQN+C) perform better than only Q-Learning on the robot. Something similar applies to hard problems. The performance gap between solving at the edge vs. the cloud is however larger, and the decision module on DEDs almost always further offloads the workload to the cloud. Since most requests in this case are handled by the cloud nodes, their performance resembles that of the CO approach.

Finally, for mixed workloads, for some problems the best option is to solve on the robot, for some problems to solve on the DEDs and for some on the cloud instances. Intelligent offloading mechanisms both on the robot and on edge devices always perform better.

5.6 Energy efficiency

To demonstrate that our architecture is capable of supporting further goals than latency minimization, we implement an energy cost-aware decision-making scheme. For this purpose, we apply a different reward model to account for the energy cost of offloading in our Q-learning-based mechanism. In particular, our (negative) reward is defined as the cost $e(p)$ of a decision (offload problem p or solve it locally) given by the following expression:

$$e(p) = \begin{cases} \text{normalised size of } p, & \text{if } p \text{ is offloaded} \\ \text{normalised hardness of } p, & \text{if } p \text{ is executed locally.} \end{cases}$$

This is an abstract, simplified model whose limitations we acknowledge. Unfortunately, the robot platform in our testbed did not provide reliable energy consumption information, so we resorted to such a model for the purpose of demonstration of potential offloading gains. We are aware that more accurate models that can account for the particularities of different connectivity and computation technologies are necessary, though this is beyond the scope of this paper.

In our setup, energy-driven decisions make more sense on the robot, as the only battery-powered device. We thus only compare the following three configurations: RO, EO, and Q+E. We do not consider energy costs on the devices with a continuous power supply, although our model can be adapted to support that. We also omit the configurations that involve cloud execution, as this is identical to DED execution from a device energy consumption perspective – the energy cost for the device is the same whether it offloads to the edge or to the cloud. Figure 4 shows a comparison of the three configurations mentioned above. In particular, it plots the percentage of energy cost saved by using a Q-learning based offloading decision on the robot vs. solving problems locally (RO) or always offloading to edge devices (EO). When using our system, the decision-making module opts for a minimum of needed energy from the local execution and offloading, resulting in lower overall energy consumption. Given our simplified energy cost model, we observe that for the simple and medium problem sets, it is generally better to solve locally, while for hard problems, it is better to offload. In most cases, the size of the problems and the complexity correlate, but it

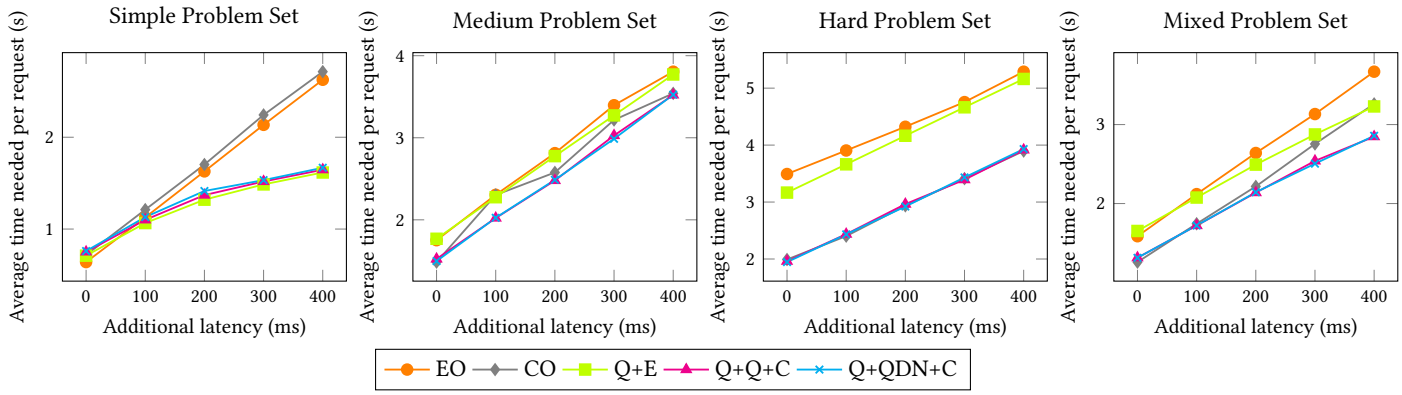


Figure 3: Average SMT request handling times for different system configurations and for different types of SMT problems. For reference, the average response times for the robot-only configuration are 5.058, 46.254, 104.176 and 46.045 s for the simple, medium, hard and mixed problem sets, respectively. These values do not depend on the latency in the path, and are omitted from the plots for reasons of readability.

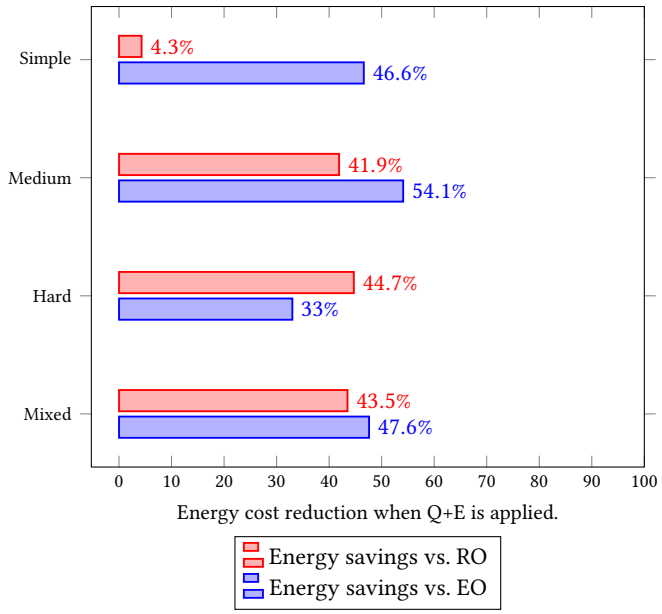


Figure 4: Comparison with energy-aware mode

is often the case that the description (the file size) is large, but the problem is not that complex (communication costs dominate) and vice versa (computation costs dominate). We note that significant gains can be achieved. For example, for the medium set, we can reduce the needed energy by more than half.

5.7 Use Case: Path Planning for Fog-Supported Robots

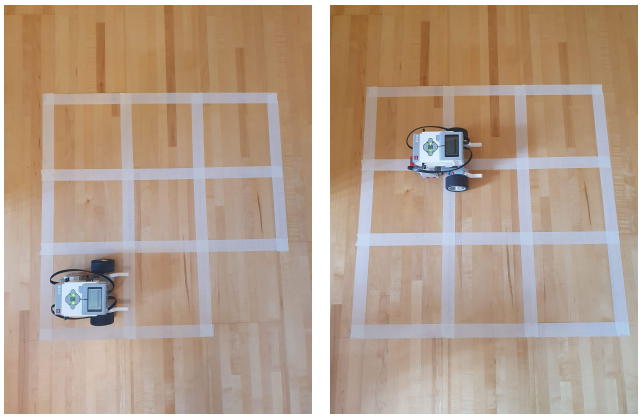
5.7.1 High-level description. In this section, we present a real use case in the field of fog robotics and illustrate a 360° end-to-end view of our system, where a problem is translated into an SMT formula, processed by our system and the result is utilized by a robot.

In particular, we consider the following problem that may emerge in contemporary Industrial IoT settings: a robot operates in a warehouse, receiving a continuous stream of commands, each containing a set of items (e.g., construction materials) to pick up and carry to its base. Each material/item is placed at a fixed location. These materials represent the vertices of a graph, and the ways between them represent the edges. The robot should not pick up materials twice, and therefore every vertex should be visited only once, leading to a Hamiltonian path problem. This scenario could be extended with more constraints, such as different edges having different costs, etc. The robot is required to end its route with the starting point again, and therefore we formulate a Hamiltonian cycle problem dynamically. In order for our system to process such commands, an application running on board the robot translates the list of items to pick to the respective SMT-LIB-encoded instance of the Hamiltonian path problem. The resulting formula is passed on to the local solver API endpoint. The solution is returned to the calling application in a transparent way as to whether execution took place locally or was offloaded. In turn, the solution is translated to a sequence of low-level instructions for the robot, so that the latter implements the derived route. The stream of commands therefore represents our SMT workload, which is possible to execute on the robot, on top of dedicated edge devices, or in the cloud, according to a decision made dynamically – on a per-command basis – by our SMT-as-a-Service framework.

5.7.2 Robot mobility. We allow the robot to move only in four directions: (i) right, (ii) left, (iii) up and (iv) down, along a grid structure with coordinates that are known to the robot. As soon as the on-board application receives a solution from the SMT-as-a-Service system, it parses the solution to obtain a sequence of vertex IDs (the path), translates the IDs to actual grid elements, and uses the Lego Mindstorms EV3 low-level API to interface with the motor in order to move the robot to the appropriate coordinates.

5.7.3 Experiment. We create two miniaturized terrain topologies to measure the performance of our scheme under planning problems of different complexity, namely on two grids with 8 and 16 vertices,

respectively, as shown in Figure 5. In the smaller grid (LHS of the figure), the robot is placed in a square representing a vertex, while in the larger one (RHS of the figure) it is the crosses of the tape that correspond to vertices. We set the resulting graphs as the input (i.e., the robot has to pick up items from all grid vertices – in both these examples a Hamiltonian cycle exists), and the SMT problem is generated by the robot as previously described. Afterwards, the on-board application invokes the local API endpoint to submit an SMT request.



(a) Small grid (8 vertices). **(b) Large grid (16 vertices).**

Figure 5: Two examples of a miniaturized grid terrain used in our experiments. The robot hosts a SMT-as-a-Service node with the Q-learning decision module, and can offload workload to a local edge cluster (not shown in the figure).

Figure 6 compares the response times for the two different grid scenarios in a testbed configuration that includes only the robot and an edge cluster of two DEDs. Solving the path planning problem on-board is costly, and therefore the robot-only (RO) approach suffers from significantly increased response times. Offloading the request is the best option, which the robot agent learns to select, and this is why the edge-only (EO) and the Q-learning (Q+E) approaches have similar performance. In more general and complex settings, where the robot application may at the same time have to solve other problems besides path planning, leading to mixed SMT formula workloads, it is reasonable to expect that a RL-based mechanism would effectively balance between on-device, edge, and cloud workload execution.

6 CONCLUSION

We presented an architecture to support the execution of SMT problem solving workloads as a service over a distributed computing infrastructure spanning the device-to-cloud continuum. Our design caters for the diversity of the continuum's ecosystem, both in terms of host capabilities, but also of supported CPS/IoT scenarios. We focused on two complementary aspects, namely those of architecture support targeting modularity, extensibility, and service orientation, and intelligent offloading decisions, showcasing the utility of reinforcement learning approaches. We have demonstrated the feasibility and performance of our SMT-as-a-Service approach in a

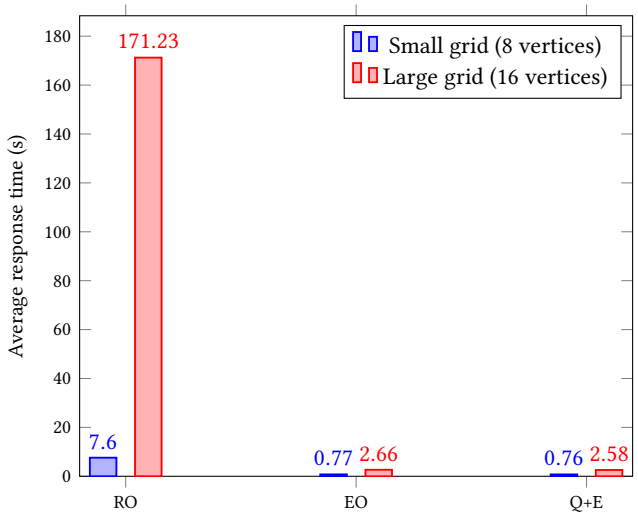


Figure 6: Path resolution time for graphs of different size. The reported results are the average of five Hamiltonian cycle problem executions.

full-fledged end-to-end testbed, with a particular view towards fog robotics. However, our design is generic and has wider applicability to any scenario where SMT finds use as a modeling and problem solving approach. Despite delivering a full end-to-end system, many aspects are still open, with more sophisticated offloading strategies topping the list.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101079214 (AIoTwin project) and by the Hellenic Foundation for Research and Innovation – Project 15706 RV4THINGS.

REFERENCES

[1] Ali J. Ben Ali, Marziye Kouroshli, Sofiya Semenova, Zakieh Sadat Hashemifar, Steven Y. Ko, and Karthik Dantu. 2023. Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping. *ACM Trans. Embed. Comput. Syst.* 22, 1 (2023), 18:1–18:31.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2017. *The SMT-LIB Standard: Version 2.6.* Technical Report. Department of Computer Science, The University of Iowa. https://www.smt-lib.org

[3] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.).

[4] Arthur Bit-Monnot, Francesco Leofante, Luca Pulina, and Armando Tacchella. 2019. SMT-based Planning for Robots in Smart Factories. In *Proc. 32nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE).*

[5] Victor Casamayor-Pujol and Schahram Dustdar. 2021. Fog Robotics - Understanding the Research Challenges. *IEEE Internet Comput.* 25, 5 (2021), 10–17.

[6] Haowei Chen, Shuiguang Deng, Hongze Zhu, Hailiang Zhao, Rong Jiang, Schahram Dustdar, and Albert Y. Zomaya. 2022. Mobility-Aware Offloading and Resource Allocation for Distributed Services Collaboration. *IEEE Trans. Parallel Distributed Syst.* 33, 10 (2022), 2428–2443.

[7] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. 2019. Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning. *IEEE Internet of Things Journal* 6, 3 (2019), 4005–4018.

[8] Youdong Chen, Qiangguo Feng, and Weisong Shi. 2018. An Industrial Robot System Based on Edge Computing: An Early Experience. In *Proc. USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*, Irfan Ahmad and Swaminathan Sundararaman (Eds.).

[9] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. 2013. The MathSAT5 SMT Solver. In *Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

[10] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, C. R. Ramakrishnan and Jakob Rehof (Eds.).

[11] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (2011), 69–77.

[12] Maria Diamanti, Panagiotis Charatsaris, Eirini-Eleni Tsiropoulou, and Symeon Papavassiliou. 2022. Incentive Mechanism and Resource Allocation for Edge-Fog Networks Driven by Multi-Dimensional Contract and Game Theories. *IEEE Open J. Commun. Soc.* 3 (2022), 435–452.

[13] Maria Diamanti, Christos Pelekis, Eirini Eleni Tsiropoulou, and Symeon Papavassiliou. 2023. Delay Minimization for Rate-Splitting Multiple Access-Based Multi-Server MEC Offloading. *IEEE/ACM Transactions on Networking* (2023). In press.

[14] Marco Gario and Andrea Micheli. 2015. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *Proc. SMT Workshop*.

[15] Guoqiang Hu, Wee-Peng Tay, and Yonggang Wen. 2012. Cloud robotics: architecture, challenges and applications. *IEEE Netw.* 26, 3 (2012), 21–28.

[16] Peng Huang, Liekang Zeng, Xu Chen, Ke Luo, Zhi Zhou, and Shuai Yu. 2022. Edge Robotics: Edge-Computing-Accelerated Multirobot Simultaneous Localization and Mapping. *IEEE Internet Things J.* 9, 15 (2022), 14087–14102.

[17] William N. N. Hung, Xiaoyu Song, Jindong Tan, Xiaojuan Li, Jie Zhang, Rui Wang, and Peng Gao. 2014. Motion planning with Satisfiability Modulo Theories. In *Proc. 2014 IEEE International Conference on Robotics and Automation (ICRA)*.

[18] Frank Imeson and Stephen L. Smith. 2019. An SMT-Based Approach to Motion Planning for Multiple Robots With Complex Constraints. *IEEE Trans. Robotics* 35, 3 (2019), 669–684.

[19] Binayak Kar, Widhi Yahya, Ying-Dar Lin, and Asad Ali. 2023. Offloading Using Traditional Optimization and Machine Learning in Federated Cloud-Edge-Fog Systems: A Survey. *IEEE Commun. Surv. Tutorials* 25, 2 (2023), 1199–1226.

[20] Vasileios Karagiannis, Pantelis A. Frangoudis, Schahram Dustdar, and Stefan Schulte. 2023. Context-Aware Routing in Fog Computing Systems. *IEEE Trans. Cloud Comput.* 11, 1 (2023), 532–549.

[21] Fahime Khoramnejad and Melike Erol-Kantarci. 2021. On Joint Offloading and Resource Allocation: A Double Deep Q-Network Approach. *IEEE Trans. Cogn. Commun. Netw.* 7, 4 (2021), 1126–1141.

[22] Nidhi Kumari, Anirudh Yadav, and Prasanta K. Jana. 2022. Task offloading in fog computing: A survey of algorithms and optimization techniques. *Comput. Networks* 214 (2022).

[23] Nancy Leveson. 2020. Are you sure your software will not kill anyone? *Commun. ACM* 63, 2 (2020), 25–28.

[24] Ji Li, Hui Gao, Tiejun Lv, and Yueming Lu. 2018. Deep reinforcement learning based computation offloading and resource allocation for MEC. In *Proc. IEEE WCNC*.

[25] Chieh-Jan Mike Liang, Lei Bu, Zhao Li, Junbei Zhang, Shi Han, Börje F. Karlsson, Dongmei Zhang, and Feng Zhao. 2016. Systematically Debugging IoT Control System Correctness for Building Automation. In *Proc. 3rd ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys@SenSys)*.

[26] Yu Liu, Yingling Mao, Zhenhua Liu, Fan Ye, and Yuanyuan Yang. 2023. Joint Task Offloading and Resource Allocation in Heterogeneous Edge Environments. In *Proc. IEEE INFOCOM*.

[27] Meysam Masoudi and Cicek Cavdar. 2021. Device vs Edge Computing for Mobile Services: Delay-Aware Decision Making to Minimize Power Consumption. *IEEE Trans. Mob. Comput.* 20, 12 (2021), 3324–3337.

[28] Claudio Menghi, Enrico Viganò, Domenico Bianculli, and Lionel C. Briand. 2021. Trace-Checking CPS Properties: Bridging the Cyber-Physical Gap. In *Proc. 43rd IEEE/ACM International Conference on Software Engineering*.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[30] Mujahid Mohsin, Zahid Anwar, Ghaith Husari, Ehab Al-Shaer, and Mohammad Ashiqur Rahman. 2016. IoTSAT: A formal framework for security analysis of the internet of things (IoT). In *Proc. 2016 IEEE Conference on Communications and Network Security (CNS)*.

[31] Zhaolong Ning, Peiran Dong, Xiaojie Wang, Lei Guo, Joel J. P. C. Rodrigues, Xiangjie Kong, Jun Huang, and Ricky Y. K. Kwok. 2019. Deep Reinforcement Learning for Intelligent Internet of Vehicles: An Energy-Efficient Computational Offloading Scheme. *IEEE Transactions on Cognitive Communications and Networking* 5, 4 (2019), 1060–1072.

[32] Zhaolong Ning, Peiran Dong, Xiaojie Wang, Joel J. P. C. Rodrigues, and Feng Xia. 2019. Deep Reinforcement Learning for Vehicular Edge Computing: An Intelligent Offloading System. *ACM Trans. Intell. Syst. Technol.* 10, 6, Article 60 (Oct. 2019).

[33] Sina Shahhosseini, Arman Anzanpour, Iman Azimi, Sina Labbaf, Dongjoo Seo, Sung-Soo Lim, Pasi Liljeberg, Nikil D. Dutt, and Amir M. Rahmani. 2022. Exploring computation offloading in IoT systems. *Inf. Syst.* 107 (2022).

[34] Richard S. Sutton and Andrew G. Barto. 2020. *Reinforcement learning - an introduction* (second ed.). MIT Press.

[35] Hoa Tran-Dang and Dong-Seong Kim. 2021. FRATO: Fog Resource Based Adaptive Task Offloading for Delay-Minimizing IoT Service Provisioning. *IEEE Trans. Parallel Distributed Syst.* 32, 10 (2021), 2491–2508.

[36] Sharda Tripathi, Corrado Puligheddu, Somreeta Pramanik, Andres Garcia-Saavedra, and Carla Fabiana Chiasserini. 2023. Fair and Scalable Orchestration of Network and Compute Resources for Virtual Edge Services. *IEEE Transactions on Mobile Computing* (2023). In press.

[37] Christos Tsigkanos, Marcello M. Bersani, Pantelis A. Frangoudis, and Schahram Dustdar. 2022. Edge-Based Runtime Verification for the Internet of Things. *IEEE Trans. Serv. Comput.* 15, 5 (2022), 2713–2727.

[38] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A. Gunter. 2019. Charting the Attack Surface of Trigger-Action IoT Platforms. In *Proc. ACM Conference on Computer and Communications Security (CCS)*.

[39] Joe Weinman. 2017. The 10 Laws of Fogonomics. *IEEE Cloud Comput.* 4, 6 (2017), 8–14.

[40] Zeinab Zabihi, Amir Masoud Eftekhari Moghadam, and Mohammad Hossein Rezvani. 2024. Reinforcement Learning Methods for Computation Offloading: A Systematic Review. *ACM Comput. Surv.* 56, 1 (2024).

[41] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. 2016. Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE Access* 4 (2016), 5896–5907.

[42] Yunzhi Zhao, Fen Hou, Bin Lin, and Yuxuan Sun. 2023. Joint Offloading and Resource Allocation With Diverse Battery Level Consideration in MEC System. *IEEE Trans. Green Commun. Netw.* 7, 2 (2023), 609–625.