Future Generation Computer Systems 142 (2023) 4–13

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Towards containerized, reuse-oriented AI deployment platforms for cognitive IoT applications

Tiago Veiga [a], Hafiz Areeb Asad [b], Frank Alexander Kraemer [b,*], Kerstin Bach [a]

[a] Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway
[b] Department of Information Security and Communication Technology, Norwegian University of Science and Technology, Trondheim, Norway

ARTICLE INFO

Article history:
Received 12 July 2022
Received in revised form 30 October 2022
Accepted 22 December 2022
Available online 24 December 2022

Keywords:
Cognitive IoT
Self-adaptive IoT
Cognitive architecture
Container-based deployment

ABSTRACT

IoT applications with their resource-constrained sensor devices can benefit from adjusting their operations to the phenomena they sense and the environments they operate in, leading to the paradigm of self-adaptive, autonomous, or cognitive IoT. On the other side, current AI deployment platforms focus on the provision and reuse of machine learning models through containers that can be wired together to build new applications. The challenge is that composition mechanisms of the AI platforms, albeit effective due to their simplicity, are in fact too simplistic to support cognitive IoT applications, in which sensor devices also benefit from the machine learning results. Our objective is to perform a gap analysis between the requirements of cognitive IoT applications on the one side and the current functionalities of AI deployment platforms on the other side. In this work, we provide an overview of the paradigms in AI deployment platforms and the requirements of cognitive IoT applications. We study a use case for person counting in a skiing area through camera sensors, and how this use case benefits from letting the IoT sensors have access to operational knowledge in the form of visual attention models. We describe the implementation of the IoT application using an AI deployment platform, analyze its shortcomings, and necessary workarounds. From the use case, we identify and generalize five gaps that limit the usage of deployment platforms: the transparent management of multiple instances of components, a more seamless integration with IoT devices, explicit definition of data flow triggers, and the availability of templates for cognitive IoT architectures and reuse below the top-level.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

In a wide range of application domains, there is an increasing number of combined Internet-of-Things (IoT) / Artificial Intelligence (AI) applications in which decisions are made by machine learning models based on the data collected by sensor devices. In principle, such applications can be implemented using a rather simple architecture, in which the concerns between IoT and AI are well separated and follow a simple, uni-directional data flow: Sensors measure a phenomenon in the environment and transmit data into a database, from which it can be analyzed to support specific decisions. In this simple division, functionality that supports sensor operations regards mainly managing their software and communication, to provide them with configurations, security or other updates and ensure their proper operation [1]. On the other hand, the main challenge of the system related to AI is the correct and efficient deployment of AI models. Here we see the move towards containerized platforms, in which ready-made AI models can be instantiated. Virtualized environments in form of containers allow to provide the correct version of the software stack for the specific models. This ensures that models run in the same environment where they were developed, tested, and validated in. Models and functions contributed by different containers can then be composed into more comprehensive data flows through connections between containers.

However, such a strict division between IoT and AI prevents systems from evolving towards higher efficiency and better performance: IoT sensors can greatly benefit from the ability to reason about their own operation and the environment they operate in so that they can reuse their constrained resources strategically. This refers to the concept of self-adaptive [2,3], context-aware, or cognitive IoT [4,5]. With increasing computational power, sensor devices can take autonomous decision to improve the system performance and actively improve their own information about the environment. For instance, they can better decide when and how to make measurements or which data to forward, to avoid spending effort on useless data with little impact on the utility for a user, also referred to as value-of-information [6]. Such behavior

* Corresponding author.
E-mail address: kraemer@ntnu.no (F.A. Kraemer).

https://doi.org/10.1016/j.future.2022.12.029
0167-739X/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

requires some form of adaptation. Due to the scale and number of devices, this adaptation must happen autonomously, and, due to the heterogeneity of the environments of the devices, also for each sensor individually.

The cognitive abilities required for such adaptive behavior are often provided through mechanisms that need similar support for machine learning as the data analysis of domain data. This implies more complex architectures that require more fine-grained management and a deeper integration with AI deployment platforms. Instead of sensors only delivering data for analysis, they are also the receivers of insights generated by AI. Such examples are models about the environment, models about the phenomena to sense or about the usage patterns, all allowing the sensor devices to plan their operations more strategically.

Such a more sophisticated coupling between IoT and AI, however, goes beyond what the simplistic composition mechanisms of current AI deployment platforms allow. In this paper we analyze the intersection between architectures for IoT cognitive models and container-based composition and deployment used in AI platforms. We present the requirements of cognitive, self-adaptive IoT applications illustrated by a case study. In particular, we study a use case where a visual sensor network with cameras is used to estimate the busyness and utilization of a skiing area. We formulate an ideal system model for a cognitive IoT solution following the reference model in [4], which emphasizes the decomposition into logical components (as opposed to only containers), and the identification of autonomous loops as well as explicit triggers. We then analyze and discuss how such solutions can be implemented following the simple paradigms for AI composition platforms based on containers as much as possible. Here we experience that the simplicity that lies in the container composition paradigm used for AI deployments is a hurdle for such an integration, and identify gaps where a straightforward implementation of the cognitive IoT principles is not possible. We therefore also suggest how current AI composition platforms can be extended to support the need for the more elaborate requirements of cognitive, self-adaptive IoT applications.

In the following, we provide an overview of related work in Section 2, and then present the principles of container-based AI solutions in Section 3. After that, we present our case study related to person counting in a skiing area in Section 4, for which we first develop a simple version to illustrate the principles of container-based deployment platforms. We then motivate the need for cognitive IoT applications in general in Section 5, and describe a specific version of a cognitive variant of the use case in Section 6. We model this cognitive version in Section 7 using a reference model as starting point, and by that identify critical requirements for the implementation on a container-based platform, which we will analyze in Section 8. In this section, we analyze the unfulfilled requirements, possible design alternatives and necessary workarounds, and propose extensions for future versions of such platforms.

2. Related work

The design of architectures for IoT networks share many architectural goals with the development of service-oriented software architectures [7]. Both share the need for efficient communication between different components and facilitate the deployment of complex networks with simple building blocks. Therefore, the microservice approach was introduced to IoT, for instance for manufacturing systems [8] or different tiers of IoT systems in general [9]. Microservices encapsulate specific, self-contained functions as loosely coupled components that communicate through message passing. Through the loose coupling, microservices support solutions based on virtualization so that the different components can execute in different environments [10]. However, it does not take into account the possibility that different devices naturally have variable computational and energy resources.

Cognitive architectures [11], on the other hand, can adapt to the current conditions in the environment and drive adaptive behavior from the system. The core feature is that such architectural designs allow data to flow arbitrarily and not only unidirectionally. Therefore, systems can adapt to what is observed from the environment and pass updated action plans back to the devices interacting with the physical world. Several concrete examples can be found in the literature, such as architectures for healthcare coaching systems [12], smart grids [13] or the management of robotic recycling plants [14]. Later, a survey analyzed several approaches for cognitive models and a general blueprint model was proposed [4].

IoT forms a three-tiered architecture, consisting of devices, edge computing resources and cloud computing [15]. In this setting, cognitive architectures can be subdivided into several components, with some of its features provided by AI services. In general, AI services can be added to an IoT network to provide additional features in the system. For example, the integration of IoT and AI services for supply chain management [16], the implementation of Edge–Cloud architectures for AI services over 5G networks [17] or the management of cloud–edge orchestration [18]. Furthermore, while edge computing can improve the quality of experience for an IoT application [19], AI can improve the implementation of secure microservices on the edge [20]. Other approaches distribute the execution of machine learning, both inference or training, over several tiers of the architecture, also referred to as Edge-to-Cloud continuum [21]. Some use containers for the execution of tasks [22,23], others distribute the computation of neural networks so that the individual layers are processed in different system tiers [24,25]. However, while these approaches clearly address the need for integration of AI and IoT, they focus on the feasibility of the approaches and the handling of constrained resources, not the composition or aspects of reuse that is starting to get mature for cloud-based solutions.

The deployment process for AI solutions is complex and, therefore, deployment platforms were presented with the goal of automating the workflow, from solution design to orchestration. One further step is the creation of community-maintained catalogs, allowing practitioners to test and reuse different components for their specific solutions. The Acumos platform [26] offers such tools, although initially conceived for unidirectional machine learning pipelines. Later, the AI4EU project launched the AI4EU platform [27] which extends Acumos to allow, among others, cyclic topologies.

In this work we aim at bridging the gap between these subtopics. In particular, we notice a lack of analysis of the deployment process of cognitive architectures for IoT applications which will benefit it generalization and easier deployment for different scenarios. Simultaneously, AI deployment platforms were not developed with the specific requirements of IoT applications, and cognitive models in particular, in mind.

3. Container-based AI deployment platforms

In this section, we provide an overview of the current status of container-based AI platforms and especially the principles they employ to make the deployment of AI pipelines less complex. Without specific support, deploying AI solutions such as machine learning models can be a difficult process, with challenges identified at every step of the deployment [28]. Deployment platforms make AI models easier deployable in various practical settings by allowing to create new solutions by wiring together reusable, off-the-shelf components. In this way, mature and tested models and processes are available for reuse and can be combined and adapted to fit into new settings and applications.

The AI4EU Experiments Platform [27], which is part of an EU project [29] and the basis for our case study, is built around two main components: the marketplace and the design studio. The marketplace is a repository for AI components and other support components, ranging from data sources to user interfaces. The design studio is a visual editor allowing users to connect different components from the marketplace catalog and create new solutions.

Inspired by development in software engineering, the basis for offering reusable components are containers. A container is a virtualized environment that runs software instances in an isolated environment from its physical host server. This allows developers to encapsulate whatever algorithms and methods are served by a given micro-service as a self-contained environment, which means freedom to use any version of the base operating system, system packages, or coding framework. Thus, using containers avoids compatibility issues in case of system upgrades or combining components that require different (and potentially incompatible) software stacks. Models can run in the exact same software stack they were originally developed, tested, and validated for.

In the AI4EU platform, a component is composed of a pointer to a Docker container available in a public or private library like for instance Docker Hub [30]. The respective micro-services and data interfaces are defined in the Protocol Buffers format [31].

The conceptual simplicity of connecting components, which sometimes only consist of linear pipelines, allows for simple graphical editing tools to design the data flows between components. This makes the deployment process easier for the system designers, as these editing tools are used to create solutions with limited knowledge about the technical details of each module. Based on the definitions of the components, the graphical editor allows the user to connect interfaces between components. The editor visually differentiates between input and output interfaces and generates similar symbols for similar interfaces such that the user can identify them. In the AI4EU platform, the orchestration for the interactions of the containers is automatically generated based on the data flow between the containers according to the wiring defined by the user in the design studio.

The modularity of these solution increases the reusability of architectures. The whole or part of the pipeline can be reused in different scenarios, for instance, experimenting with a pipeline of different AI modules or databases while maintaining the overall structure design. In case a solution cannot be realized by wiring existing components, users can also create own components with arbitrary logic. Alternatively, the system designer can define the orchestration links programmatically or implement a custom orchestration module, both alternatives that require considerable programming expertise.

To sum up, AI deployment platforms here represented with the AI4EU platform allow to create AI solutions by offering components as self-contained environments through containers. These components can be combined using graphical editors.

4. Use case: Person counting in a skiing area

Our case study is a system that estimates the business of a skiing area. It processes camera images and counts the number of persons as basis for the estimation. For this simple initial version, the design within the AI4EU editor is straightforward, with only a few components and connectors. Fig. 1 shows the first basic workflow of the system for a single camera instance. The pipeline begins with a data source component, which fetches the images published by the camera. Then, an image recognition component follows. This component is reused from the publicly available components in the AI4EU platform catalog. It encapsulates an instance of an off-the-shelf, reusable module for object detection that follows the *You only look once* approach (YOLO, [32]), which allows for the detection of several different objects in various positions in a single inference step. The model's output is a list of bounding boxes, illustrated in Fig. 2. The bounding boxes are annotated with the type of object detected and a confidence level. Finally, the image and list of detections are passed to a dashboard component, which implements a web user interface to allow real-time visualization.

5. Towards self-adaptive, cognitive IoT

Before we transform our simple use case from above into a cognitive version, we first motivate why we need cognitive applications in IoT at all. To make the deployment of IoT devices cheaper or feasible at all, it is crucial that they are wireless, use a wireless protocol for communication, and are powered by batteries or preferably by energy harvesting to reduce the need for manual maintenance. It is also desirable to keep devices as compact as possible to reduce costs and make them less obtrusive, reducing the size of harvesters such as solar panels and energy buffers. Energy is a scarce resource that should be used strategically [33]. Other relevant constraints are the transmission over wireless channels, which may have restrictions on bandwidth.

In IoT, various techniques are employed to handle resources economically, such as low-power electronics, efficient communication protocols [34], new types of energy buffers [35], and energy harvesting [36]. Another class of approaches tries to maximize the utility of the system to the user by minimizing the effort spent on data that is not significant. These can be summarized by the concept of Value of information [6]. For the control software in devices, this means selecting more carefully when and where to make measurements, and which data to process, transmit or store.

So, in general, the idea is to only use resources on relevant data for the system's utility. Such knowledge about the value of information is often highly specific to the application goals, the phenomena to sense, the sensor's environment, and the specific construction of the device. The knowledge may hence vary for each individual sensor device, and can vary over time as the environment and the sensed phenomena can be non-stationary. In general, this means that IoT sensors themselves benefit from learning processes so that they can more optimally control their operation and make tradeoffs between, for instance, energy consumption and utility to the user to optimize overall operation. They may also be able to degrade their level of service gracefully, or manage to maintain a minimum service level [37] when resources get scarce.

Many of the reasoning techniques for these systems involve machine learning [4]. For the connection between IoT and AI this means that the IoT devices of a system are not only used as a source of data that is analyzed and acted upon through AI, but that AI solutions also provide feedback to the IoT devices to learn about their own properties and the environment they are deployed in, with the aim to optimally control their own operation. Systems hence evolve from a uni-directional, simplistic pipeline into more general cognitive models with learning and reasoning processes, and the ability to adapt.

We focus in the following on the effect of such a paradigm shift in IoT on system development with the starting point of the containerized platform that support current AI solutions, and ask how they can be extended so that they fulfill the requirements of cognitive IoT.

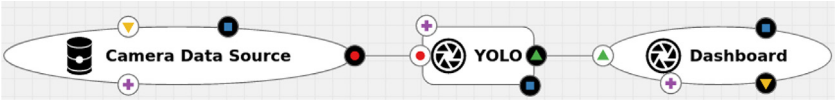

Fig. 1. Screenshot of the solution designed in the AI4EU platform for a unidirectional pipeline for the simplified use case without any self-adaptation or feedback to the sensor devices.

Fig. 2. Two persons detected by the YOLO image recognition model with corresponding confidence.

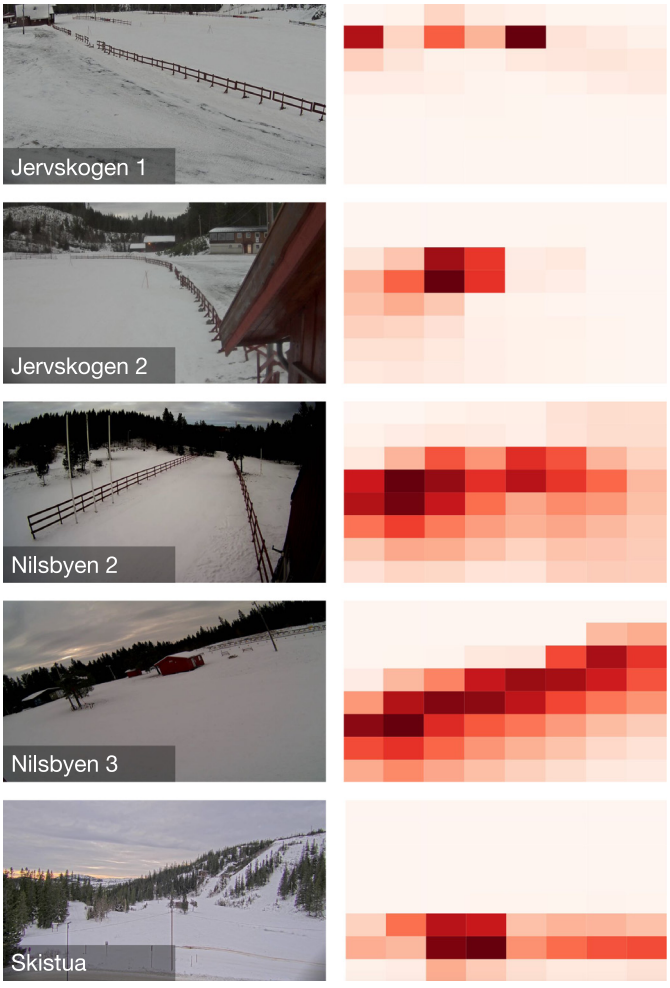

Fig. 3. Camera views together with their visual attention models, calculated over the entire range of images captured. The darker certain cells in the attention model are, the more often persons have appeared and therefore should be prioritized during transmission.

Fig. 4. Snapshots of the visual attention model of one camera over time (day to day). This shows how sensor devices can acquire knowledge over time that they can exploit to reduce image transmissions.

6. Use case: Self-adaptive, cognitive version

To enable the sensors to better adapt to constrained resources, we will extend the system into a cognitive, self-adaptive IoT system in which sensors can benefit from the system's feedback to adjust their operation. Hence, AI functions integrate also directly with the operation of the system. For visual sensing networks, there exists a variety of techniques to make operations more efficient. In the following, we focus on a concept referred to as *visual attention* [38]. When revisiting Fig. 2, we see several parts in the system where it is unlikely to observe persons, like in the sky. When transmitting an image in constrained situations, sensor devices could hence drop the transmission of those parts of an image that are less likely to show persons. Compared to always sending the complete image and then running out of energy, this solution would let the system maintain its utility and still count the number of persons. We therefore partition an image into a set of $N = 64$ tiles. With $V[t]$, $t \in \{1..N\}$ we store a visual attention model. The higher the value for each tile, the more likely it is that persons appear in this part of the image. The logic consists then of two parts:

- On the server, where the object detection runs using the YOLO model just as before, we also create a visual attention model for each camera. This happens by analyzing the position of the bounding boxes delivered by the object detection and periodically updating the attention model. The server regularly transmits the attention model to the sensor device.
- On the sensor device, we separate the image into tiles and select which tiles to transmit to the server using a selection policy. This policy determines the number of tiles to send using a planning algorithm depending on its current energy budget and then transmits only those tiles deemed valuable by the attention model.

For this to work, the server receives the transmitted tiles, stitches them together with a background image received earlier, and then performs the object detection on these reduced tile sets. The details of the algorithms for the computation of the attention models and the policies are published in [39].

Fig. 3 shows five cameras and their corresponding attention models as heatmaps to the right. The cells in the heatmap with darker coloring correspond to those areas where the detection of persons is more likely. Fig. 4 shows the visual attention map of one camera as it develops over time, from day to day.

To evaluate our approach, we simulated five different policies for each device, which vary in the number of tiles transmitted. The fewer tiles transmitted, the lower the usage of energy resources. Fig. 5 shows the results of different simulation runs. The

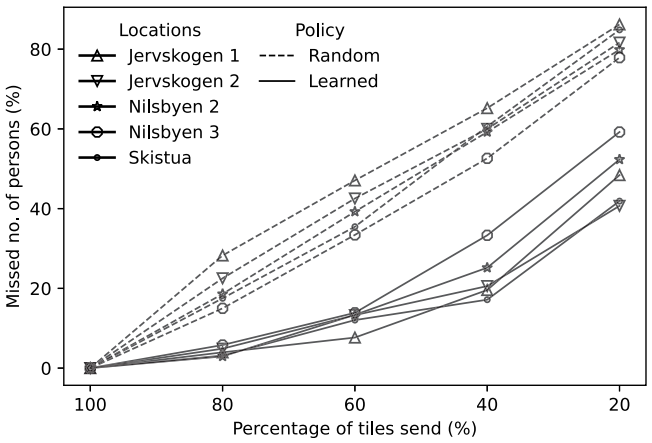

Fig. 5. Percentages of missed person detections (vertical axis) for each camera for the five tile percentage policies (horizontal axis). Results show the visual attention models (solid lines) and random policies (dashed lines). On average, using the attention models reduces the number of undetected persons by 55%.

horizontal axis shows different percentages of tiles sent, starting with 100% (all tiles) to only 20% of the tiles. The vertical axis shows the percentage of persons that remained undetected. Since we took the number of detected persons in the complete images as ground truth, the error and hence the number of undetected persons is 0 for the policy transmitting 100% of the tiles. The solid lines represent the policies that make use of the visual attention models. We see that the percentages of undetected persons rise as we transmit fewer tiles, which is not unexpected.

When comparing the different policies using the visual attention models (solid lines) compared to random policies (dashed lines), we see that the ones using visual attention models perform better and benefit from the cognitive architecture and self-adaptive model. The number of undetected persons using the random tile selection is much higher. When we average over all cameras and transmission levels, we observe that the visual attention models reduce the number of undetected persons by 55% [39]. The exact tradeoff between energy usage and accuracy through the selection of transmission levels is not in focus here, but the error reduction compared to random policies shows the potential of a cognitive, self-adaptive approach to IoT.

7. Cognitive IoT model

We will now develop the implementation of the cognitive version of the use case as motivated and outlined above. It will serve as a source for the requirements we explore that should be supported by containerized AI deployment platforms. The self-adaptive, cognitive version of the application will turn out more complex than the basic version from Section 4 and Fig. 1, as it contains more data flows, includes feedback loops, and also needs to learn, maintain and distribute the visual attention models. We approach its implementation following the general cognitive model for IoT applications introduced by Braten et al. [4]. This model generalizes patterns for adaptive behavior commonly found in different architectures surveyed in the literature on autonomous computing and self-adaptive systems. Since the reference architecture was extracted from case studies in the literature, it applies to a wide range of systems, which means that the requirements we formulate based on the cognitive version of our use case should also generalize well for other applications.

The main elements of the reference models are components, loops, and triggers. Components determine the locus of computation and encapsulation and are relevant for organization and potential for reuse, the loops organize the data flow through the system, and the triggers determine *when* data flows are dispatched and computation happens. Fig. 6 illustrates the design of this cognitive architecture, which we will describe in the following.

7.1. Component structure

Components are classified according to their function in the cognitive model. They can encapsulate declarative **(DK)** or procedural knowledge **(PK)**, the acquisition or measurement of data and hence responsible for perception **(P)**, execute actions **(A)**, or be part of the adaptation process **(AP)**, following the taxonomy of autonomous systems.

The sensor devices (to the left in Fig. 6) are structured by two components. *Perceive* is responsible for acquiring images and their subsequent processing. *Execute* is responsible to interpret the visual attention model and decide when, how many and which tiles to send, as described in Section 6.

To the right are the device managers usually executed on a cloud platform. Here, each sensor device instance has its own instance of a manager. The (partial) images sent by the device are stored as part of the device-specific knowledge component. A monitor device examines the incoming data and dispatches it to the learning process for the visual attention models. The image recognition task is implemented within this component. These models are handed over to the analytical component that predicts device operation and the subsequent planning component. Currently, the visual attention models are the only information provided to the devices. More advanced solutions applying planning to optimize for the available energy can improve the performance and autonomy of IoT devices further. Component *Learn* observes the learning process of the visual attention maps.

7.2. Loops for autonomous behavior, learning and adaptation

The reference model identifies three different types of loops that describe data flows with different purposes to support (1) autonomous behavior, (2) learning, and (3) adaptation.

- **Autonomic loops** are the control mechanism that allow devices, to some degree, to take autonomous decisions independent of external information. In Fig. 6 this is loop **L1**. This loop is contained within the device and consists of the data flow between the *Perceive* and *Execute* components. This loose coupling between the device manager and the device allows the device some degree of autonomy, and it can operate even if the device manager fails to send a refreshed visual attention model.
- **Learning loops** control the update of the system's knowledge about the environment and the state of its devices. In Fig. 6 this is loop **L2**. It controls the learning process for the visual attention model and is contained within the server. The object detection container acts as a monitor that observes the sensed event (a reconstructed image) and returns new knowledge. Other learning processes can occur in parallel like, for example, the temporal model of presence of persons for each hour of each week day.
- **Adaptive loops** control reasoning mechanisms and ensure that the devices respond well to changes in the environment or, in other words, that the plans followed by the autonomic loop adapt to the observations perceived from the environment. In our application in Fig. 6, L3 is an adaptive loop that controls the transfer of an updated visual attention model to the device. It is triggered by a sensing event on the device, which is communicated to the manager which,

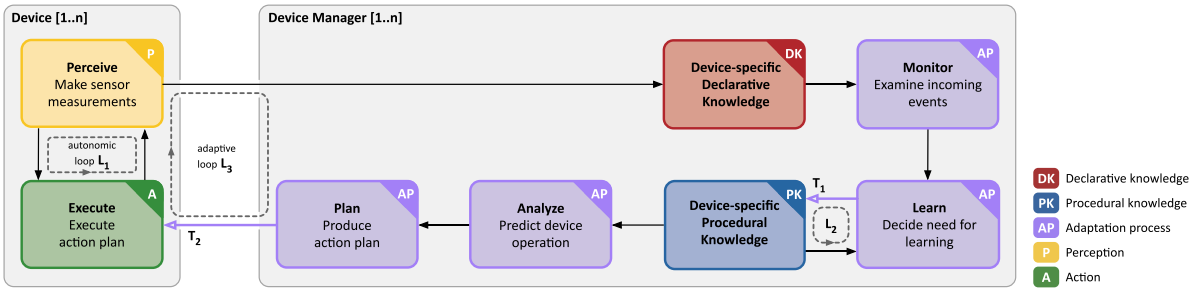

Fig. 6. Diagram with the proposed cognitive architecture for the camera deployment use case, following the reference architecture in [4]. Each device instance (left) is represented by its own device manager instance (right). The data flows between the logical components are organized by three main loops for autonomous behavior, learning and adaptation.

in turn, can decide to transfer an updated model after the learning process back to the device. This corresponds to an update of the action plan for the device and is an adaptation of the device operation.

Each loop follows the data flow interaction between different sub-parts of the network. In particular, L1 is contained within the device, L2 is within the cloud platform, and L3 involves communication between the device and the device manager in the cloud.

7.3. Triggers

Tasks such as learning are often resource-intensive and should only be dispatched when necessary. The best practices in [4], therefore, recommend making triggers explicit. We identify two critical triggers in the architecture:

- **T1** triggers the learning process. When sufficient new data is available, the learning component triggers the exchange of this information to the knowledge component to update the visual attention model. Here the formulation of the trigger influences the tradeoff between computational costs and the currentness of the visual attention model. Models can be updated with each received image, or it may be sufficient to only update the attention model once a day.
- **T2** triggers the process of transmitting an updated visual attention model to the device. It can happen either at every update or when the planning component detects that it is significantly different from the previous version.

We should note that the original reference model in [4], in addition to the device managers, also contains elements for adaptive behavior at the system level for concerns relevant for all devices. We left these out for brevity as they do not change the fundamental requirements we want to discuss here.

8. Container-based implementation

We now proceed with the implementation of the cognitive, self-adaptive model for the IoT application of Fig. 6. Of course, we want to realize as many of the benefits that come with the principles of reuse-oriented assembly of virtualized containers (described in Section 3), so that more complex IoT applications can be easily built and deployed. This means in particular creating data pipelines among several containers, reusing existing containers whenever possible, and facilitating the orchestration of the system.

However, we experienced that the current platforms lack features which prevents a straightforward deployment of our modular cognitive architecture. In this section, we discuss the main gaps and missing features, analyze our workarounds and propose improvements for the next generation of deployment platforms.

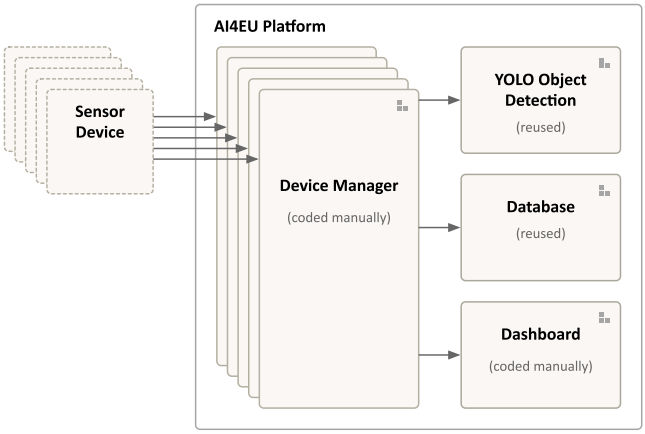

Fig. 7. Container structure for the deployed solution. Sensor devices (left) are not part of the deployment platform. Each sensor is represented by its own device manager container. Containers for the YOLO object detection and database are directly reused from the AI4EU catalog.

Fig. 7 shows the container structure of the solution built for the deployment platform. To the left are the sensor devices, which are not part of the containerized platform. The logical components within the device manager of Fig. 6 are mapped into corresponding manager containers. These manager containers act as orchestrators that connect the external devices with the other modules in the cloud network. We implemented one container instance per device instance. Three more containers provide additional functionality:

- *YOLO Object Detection* processes the images and finds the bounding boxes of persons. Like in the simplified version of the system in Fig. 1, this container was directly reused from the AI4EU catalog.
- *Database* implements and manages a MongoDB database to allow the system to look for previous information and an external user to inspect the database if necessary. Also this component could be directly reused.
- *Dashboard* exposes a web interface with information about the current status of the system for an external user, which include the current status of the detection system with a print of the current image, with detected objects highlighted, and the current attention map learnt by the system.

As indicated above, a direct deployment of this solution on the platform is impossible as it lacks support for some features required by the model in Fig. 6. We instead created and configured

Table 1
Overview of the requirements for a cognitive version of the AI platform, together with design alternatives (if available) and design choices.

Requirements	Design alternatives	Design choices	Sect.
R1: Container Instance Management We need to manage multiple instances of a component, for instance to offer one manager component instance per device.	**A1.1:** One container instance per device **A1.2:** A single container for all device instances **A1.3:** A clustered solution.	We selected A1.1 as workaround, since A1.2 and A1.3 are not supported. For future versions of the platform, A1.3 would offer the best solution for developers.	8.1
R2: Closer Integration with Devices External devices should be represented in the graphical editor.	**A2.1:** Contain device nodes in the visual editor. **A2.2:** Add generic interfaces for incoming data from devices.	We selected A2.2 as workaround with a special component as interface to the devices.	8.2
R3: Explicit and Expressive Triggers We need to define triggers explicitly.	–	As a workaround, we implemented triggers as part of the manually written logic, not visible in the graphical editor.	8.3
R4: Reuse Below the Top-Level Reuse of containers also at the sub-level, not only top-level.	–	As a workaround, we placed the container for image detection at the top level, and routed communication from the device managers to it, instead of including it inside the device manager.	8.4
R5: Architecture Templates It should be possible to reuse also templates of architectures.	–	As workaround, we constructed our model from scratch, without reusing any template.	8.5

an orchestration solution through manual programming, circumventing some of the constraints of the deployment platforms and its visual editor. The main unfulfilled requirements we identified are:

- **R1** The platform currently only offers single instances of components and does not offer a mechanism for the management of multiple instances. This is necessary for device managers, for instance, as they ideally exist as one instance per sensor device instance. (Section 8.1)
- **R2** The visual editor does not have explicit mechanisms to include external devices, but only covers the deployment in the cloud network. Instead, it should offer the possibility to also include a representation of the devices. (Section 8.2)
- **R3** It is not possible to define triggers, as identified in the cognitive model, inside the editor. Instead, these should be modeling elements within the editor, as they represent critical concepts for the execution. (Section 8.3)
- **R4** The reuse of containers is available at the top level. This considerably limits the flexibility for structuring solutions and hampers reuse. In addition, it should be possible to edit hierarchical designs and introduce also reuse for subcomponents. (Section 8.4)
- **R5** It should be possible to also allow the reuse of architecture templates. (Section 8.5)

Table 1 provides an overview of the requirements and the potential design alternatives. Since these requirements are not specific to our use case but are relevant for also other cognitive IoT applications, we discuss in the following design options, how we solved the requirements by workarounds, and how the deployment platforms can support these requirements in future versions.

8.1. R1: Container instance management

Managing many devices in a single model is a central aspect of cognitive architectures. In particular, there is device-specific knowledge that needs to be maintained, updated, and pushed down to the respective device. A manager component should therefore represent each physical device [4]. It should also be possible to create and destroy such instances on-demand, as new physical sensor devices can join or leave the system during runtime.

In principle, there are three alternatives for the implementation of device manager instances: **(A1.1)** one container per device instance; **(A1.2)** a single container handling all device instances; or **(A1.3)** a clustered solution with several containers, each handling a cluster of devices.

We used the first option as a workaround and created one container instance per physical device, since this resulted with the given restrictions in the best overview for the model. For that, we configured access ports to each container instance to ensure that each device connects to its respective manager. For our prototypical system with few cameras, this is a viable solution, but for systems with a high number of device instances this may require too many resources. Containers are meant to provide independent execution environments. Providing one execution environment for each sensor device instance is not necessary since they will probably require the same software stack anyway. Hence, options **(A1.2)** and **(A1.3)** are more suitable. In these cases, the data flows between containers need to carry also the device ID so that the containers implementing device managers for several device instances can tell them apart.

Ideally, the management of several instances should be transparent to the developer, as the required logic adds a lot of complexity without adding any application-specific value. Such a transparent management of device manager instances could also be an opportunity to perform load balancing; the deployment platform could not only create the instances when they are required, but also decide where to execute them.

8.2. R2: Closer integration with devices

Another core aspect of IoT architectures is the seamless integration of data exchanged between sensor devices and managing components. Therefore, it is desirable that the graphical platform editor offers functionalities to easily deploy solutions that cover both the cloud network and data exchange with external devices. It does not require the system designer to manually create components to implement this interface for each use case.

One option **(A2.1)** is to explicitly include the device nodes in the visual editor. For system designers, it would appear like constructing a single, coherent architecture, with the possibility

to indicate where each component is intended to run. In this alternative, it needs to be considered that most IoT devices cannot run containers (due to their limited processing power) and that a single configuration output would not be sufficient. Therefore, separate configuration files and scripts could be automatically produced by the platform for each of the network's physical components, separating code for devices from code for containers running on servers.

Another option **(A2.2)** is to offer a generic interface for incoming data that can act as a data source module. Currently, one can encapsulate data fetching scripts for specific scenarios and include those as specific components for particular use cases, similar to the container *Camera Data Source* we used in Fig. 1. This solution is neither reusable nor generic; therefore, we should aim for a more general functionality that can support the most used IoT data platforms. That way, users could have a generic interface to interact with different data sources and, possibly, with data incoming from different data platforms. Since solution **(A2.1)** requires considerable extensions of the platform, we have here opted for **(A2.2)**.

8.3. R3: Explicit and expressive triggers

As explained in Section 7.3, correct triggering is crucial for the system's efficient operation, as triggers define when data flows and computation and communications are scheduled. When triggers execute too often, computational resources may be wasted on data with no or minor changes, and when triggers are executed not often enough, information may be outdated. Moreover, in cognitive systems, triggers are not necessarily tied exclusively to events from the environment, but can also depend on the state of knowledge or the recognition of trends or other situations. Triggers are therefore an important design element in a cognitive system that deserves careful consideration.

A critical trigger in our use case is the retraining of the visual attention model. In our implementation, we update the model with every reception of a new image, which is simple but implies also unnecessary computational resources and hence energy. A better solution would be to update the model less frequently, for instance, once a day. We should hence be able to define more expressive triggers, for instance by defining minimum and maximum frequencies, depending on learning progress. Triggers should be able to take external events into account but also be independent from them if the environment suddenly does not issue such events anymore.

As a workaround, we implemented the triggers as part of the manually written containers. However, this solution is neither generic nor flexible as it is hard-coded and hidden inside containers. We suggest therefore that the visual editor allow to directly define triggers that can be connected to data flows and interact with the components.

8.4. R4: Reuse below the top-level

In the deployment process of an architecture there are typically generic and specific components. Generic components can be reused in different pipelines, often across users and project, ranging from data management (e.g., data collators that merge incoming data from different containers to a single interface) to data analysis services (e.g., containers encapsulating object detection algorithms which are provided as a service by the respective container interface). In contrast, specific components are tailored and need to be built from scratch for specific needs of each use case (e.g., dashboard containers that show specific information from the system required by a client).

In the current version of the AI4EU platform, reuse can only happen at the level of containers and only at the top-level of the system. Therefore, the *YOLO* container for object detection is currently placed at the top-level, though it would be better placed within each manager instance. As a workaround, we placed it at the top level and routed the communication from inside the device manager to it, instead of keeping this internal to the device manager.

8.5. R5: Architecture templates

In addition to the possibility of reusing containers at several levels, we suggest the possibility of offering templates for cognitive architectures. This works as a compromise between the specific needs of each component and the generic property of the reference model and as a mean to better categorize containers available in the public catalogs. In contrast to a container that can be reused, a template would allow some components to be open slots, into which specific components are placed once the template is instantiated. It would hence encapsulate interaction patterns between components and useful to document architecture traits that go beyond single cohesive components.

9. Conclusions

We analyzed the current status of deploying solutions for cognitive, self-adaptive IoT applications, in which AI is not only used to analyze domain data but also aspects of the IoT devices to facilitate optimized operations through adapting to the environment. Our work shows the effectiveness of the cognitive architecture for a specific use case, namely the detection and counting of persons in a skiing area through cameras. The operation is made more efficient using visual attention models that were learned.

AI platforms allow the user to create pipelines where components are based on containers reusing proven AI solutions. This makes them attractive to be used in such cognitive IoT solutions. However, our experience in this use case also shows that the current status of deploying platforms lack some flexible and usable features to support these cognitive architectures fully. The main gaps we identified are related to managing multiple instances of the same container, integrating IoT devices in the deployment process, including explicit triggers while creating solutions in the platforms' editing tools, and the availability of the template architectures in the platforms. For each of them, we analyze why they are important features and discuss possible suggestions for future developments of the platforms.

Deployment platforms have a high potential to facilitate the deployment process for practitioners and simplify the adoption of new solutions with integrated AI components. In particular, they easily integrate most of the core features of IoT cognitive architectures. Our analysis can help drive the development of these platforms, allowing them to cover a broader range of solutions. These developments should be made general so that they become a default offer by the platforms and then generalize for each specific use case.

CRediT authorship contribution statement

Tiago Veiga: Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Hafiz Areeb Asad:** Validation, Visualization, Formal analysis, Writing – original draft, Writing – review & editing. **Frank Alexander Kraemer:** Conceptualization, Writing – original draft, Supervision, Writing – review & editing. **Kerstin Bach:** Conceptualization, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This work was partially funded by the European Union's Horizon 2020 research and innovation program, project AI4EU, grant agreement No. 825619.

Data availability

Data will be made available on request.

References

[1] S. Sinche, D. Raposo, N. Armando, A. Rodrigues, F. Boavida, V. Pereira, J.S. Silva, A survey of IoT management protocols and frameworks, IEEE Commun. Surv. Tutor. 22 (2) (2020) 1168–1190, http://dx.doi.org/10.1109/COMST.2019.2943087.

[2] H. Muccini, M. Sharaf, D. Weyns, Self-adaptation for cyber-physical systems: a systematic literature review, in: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2016, pp. 75–81.

[3] I. Alfonso, K. Garcés, H. Castro, J. Cabot, Self-adaptive architectures in IoT systems: a systematic literature review, J. Internet Serv. Appl. 12 (1) (2021) 1–28.

[4] A.E. Braten, F.A. Kraemer, D. Palma, Autonomous IoT device management systems: Structured review and generalized cognitive model, IEEE Internet Things J. 8 (6) (2021) 4275–4290, http://dx.doi.org/10.1109/JIOT.2020.3035389.

[5] B. Athamena, Z. Houhamdi, Cognitive and autonomic IoT system design, in: 2021 Eighth International Conference on Software Defined Systems, SDS, 2021, pp. 1–7, http://dx.doi.org/10.1109/SDS54264.2021.9732121.

[6] F. Alawad, F.A. Kraemer, Value of Information in Wireless Sensor Network Applications and the IoT: A Review, IEEE Sens. J. 22 (10) (2022) 9228–9245, http://dx.doi.org/10.1109/jsen.2022.3165946.

[7] B. Butzin, F. Golatowski, D. Timmermann, Microservices approach for the internet of things, in: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation, ETFA, 2016, pp. 1–6, http://dx.doi.org/10.1109/ETFA.2016.7733707.

[8] K. Thramboulidis, D.C. Vachtsevanou, A. Solanos, Cyber-physical microservices: An IoT-based framework for manufacturing systems, in: 2018 IEEE Industrial Cyber-Physical Systems, ICPS, 2018, pp. 232–239, http://dx.doi.org/10.1109/ICPHYS.2018.8387665.

[9] C.J.L. de Santana, B. de Mello Alencar, C.V.S. Prazeres, Reactive microservices for the internet of things: A case study in Fog computing, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1243–1251, http://dx.doi.org/10.1145/3297280.3297402.

[10] M. Alam, J. Rufino, J. Ferreira, S.H. Ahmed, N. Shah, Y. Chen, Orchestration of microservices for IoT using docker and edge computing, IEEE Commun. Mag. 56 (9) (2018) 118–123, http://dx.doi.org/10.1109/MCOM.2018.1701233.

[11] C. Savaglio, G. Fortino, Autonomic and cognitive architectures for the internet of things, in: G. Di Fatta, G. Fortino, W. Li, M. Pathan, F. Stahl, A. Guerrieri (Eds.), Internet and Distributed Computing Systems, Springer International Publishing, Cham, 2015, pp. 39–47.

[12] A. Amato, A. Coronato, An IoT-aware architecture for smart healthcare coaching systems, in: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications, AINA, 2017, pp. 1027–1034, http://dx.doi.org/10.1109/AINA.2017.128.

[13] Y.C. Pranaya, M.N. Himarish, M.N. Baig, M.R. Ahmed, Cognitive architecture based smart grids for smart cities, in: 2017 3rd International Conference on Power Generation Systems and Renewable Energy Technologies, PGSRET, 2017, pp. 44–49, http://dx.doi.org/10.1109/PGSRET.2017.8251799.

[14] O.G. Rosado, P.F.M.J. Verschure, Distributed adaptive control: An ideal cognitive architecture candidate for managing a robotic recycling plant, in: V. Vouloutsi, A. Mura, F. Tauber, T. Speck, T.J. Prescott, P.F.M.J. Verschure (Eds.), Biomimetic and Biohybrid Systems, Springer International Publishing, Cham, 2020, pp. 153–164.

[15] J. Zhang, D. Tao, Empowering Things With Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things, IEEE Internet Things J. 8 (10) (2021) 7789–7817, http://dx.doi.org/10.1109/jiot.2020.3039359.

[16] G. Kousiouris, S. Tsarsitalidis, E. Psomakelis, S. Koloniaris, C. Bardaki, K. Tserpes, M. Nikolaidou, D. Anagnostopoulos, A microservice-based framework for integrating IoT management platforms, semantic and AI services for supply chain management, ICT Express 5 (2) (2019) 141–145, http://dx.doi.org/10.1016/j.icte.2019.04.002.

[17] G. Myoung Lee, T.-W. Um, J.K. Choi, AI as a microservice (AIMS) over 5G networks, in: 2018 ITU Kaleidoscope: Machine Learning for a 5G Future, ITU K, 2018, pp. 1–7, http://dx.doi.org/10.23919/ITU-WT.2018.8597704.

[18] Y. Wu, Cloud-edge orchestration for the internet of things: Architecture and AI-powered data processing, IEEE Internet Things J. 8 (16) (2021) 12792–12805, http://dx.doi.org/10.1109/JIOT.2020.3014845.

[19] G. Premsankar, M. Di Francesco, T. Taleb, Edge computing for the internet of things: A case study, IEEE Internet Things J. 5 (2) (2018) 1275–1284, http://dx.doi.org/10.1109/JIOT.2018.2805263.

[20] F. Al-Doghman, N. Moustafa, I. Khalil, Z. Tari, A. Zomaya, AI-enabled secure microservices in edge computing: Opportunities and challenges, IEEE Trans. Serv. Comput. (2022) 1, http://dx.doi.org/10.1109/TSC.2022.3155447.

[21] D. Rosendo, A. Costan, P. Valduriez, G. Antoniu, Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review, J. Parallel Distrib. Comput. 166 (2022) 71–94, http://dx.doi.org/10.1016/j.jpdc.2022.04.004, arXiv:2205.01081.

[22] S. Wang, Y. Hu, J. Wu, KubeEdge.AI: AI platform for edge devices, arXiv, 2020, http://dx.doi.org/10.48550/arxiv.2007.09227.

[23] O. Debauche, S. Mahmoudi, S.A. Mahmoudi, P. Manneback, F. Lebeau, A new Edge Architecture for AI–IoT services deployment, Procedia Comput. Sci. 175 (2020) 10–19, http://dx.doi.org/10.1016/j.procs.2020.07.006.

[24] S. Teerapittayanon, B. McDanel, H. Kung, Distributed Deep Neural Networks over the Cloud, the Edge and End Devices, in: 2017 IEEE 37th International Conference on Distributed Computing Systems, ICDCS, 2017, pp. 328–339, http://dx.doi.org/10.1109/icdcs.2017.226.

[25] F. Zhu, B.C. Ooi, C. Miao, H. Wang, I. Skrypnyk, W. Hsu, S. Chawla, A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, Y. Zhang, Auto-Split: A General Framework of Collaborative Edge-Cloud AI, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 2543–2553, http://dx.doi.org/10.1145/3447548.3467078.

[26] S. Zhao, M. Talasila, G. Jacobson, C. Borcea, S.A. Aftab, J.F. Murray, Packaging and sharing machine learning models via the acumos AI open platform, in: 2018 17th IEEE International Conference on Machine Learning and Applications, ICMLA, 2018, pp. 841–846, http://dx.doi.org/10.1109/ICMLA.2018.00135.

[27] P. Schüller, J.P. Costeira, J. Crowley, J. Grosinger, F. Ingrand, U. Köckemann, A. Saffiotti, M. Welss, Composing complex and hybrid AI solutions, 2022, http://dx.doi.org/10.48550/ARXIV.2202.12566.

[28] A. Paleyes, R.-G. Urma, N.D. Lawrence, Challenges in deploying machine learning: A survey of case studies, ACM Comput. Surv. 55 (6) (2022) http://dx.doi.org/10.1145/3533378.

[29] AI4EU, Europe's AI-on-demand platform, 2022, URL https://www.ai4europe.eu. (Last Accessed July 2022).

[30] Docker, Docker hub, 2022, URL https://www.docker.com/products/docker-hub/. (Last Accessed July 2022).

[31] Google, Protocol buffers, 2022, URL https://developers.google.com/protocol-buffers/. (Last Accessed July 2022).

[32] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, J. Fang, imyhxy, K. Michael, Lorna, V. Abhiram, D. Montes, J. Nadar, Laughing, tkianai, yxNONG, P. Skalski, Z. Wang, A. Hogan, C. Fati, L. Mammana, AlexWang1900, D. Patel, D. Yiwei, F. You, J. Hajek, L. Diaconu, M.T. Minh, ultralytics/yolov5: v6.1 - TensorRT, TensorFlow edge TPU and OpenVINO export and inference, 2022, http://dx.doi.org/10.5281/zenodo.6222936.

[33] H. Jayakumar, K. Lee, W.S. Lee, A. Raha, Y. Kim, V. Raghunathan, Powering the internet of things, in: 2014 IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED, 2014, pp. 375–380, http://dx.doi.org/10.1145/2627369.2631644.

[34] W. Ayoub, A.E. Samhat, F. Nouvel, M. Mroue, J.-C. Prévotet, Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs Standards and Supported Mobility, IEEE Commun. Surv. Tutor. 21 (2) (2019) 1561–1581, http://dx.doi.org/10.1109/COMST.2018.2877382.

[35] X. Shen, J. Tuck, R. Bianchini, V. Sarkar, A. Colin, E. Ruppel, B. Lucia, A reconfigurable energy storage architecture for energy-harvesting devices, ACM SIGPLAN Not. 53 (2) (2018) 767–781, http://dx.doi.org/10.1145/3173162.3173210.

[36] F.K. Shaikh, S. Zeadally, Energy harvesting in wireless sensor networks: A comprehensive review, Renew. Sustain. Energy Rev. 55 (2016) 1041–1054.

[37] R. Ahmed, B. Buchli, S. Draskovic, L. Sigrist, P. Kumar, L. Thiele, Optimal power management with guaranteed minimum energy utilization for solar energy harvesting systems, ACM Trans. Embedded Comput. Syst. 18 (4) (2019) 30, http://dx.doi.org/10.1145/3317679.

[38] A. Borji, L. Itti, State-of-the-art in visual attention modeling, IEEE Trans. Pattern Anal. Mach. Intell. 35 (1) (2013) 185–207, http://dx.doi.org/10.1109/TPAMI.2012.89.
[39] H.A. Asad, F.A. Kraemer, K. Bach, C. Renner, T.S. Veiga, Learning attention models for resource-constrained, self-adaptive visual sensing applications, in: Proceedings of the Conference on Research in Adaptive and Convergent Systems, 2022, pp. 165–171, http://dx.doi.org/10.1145/3538641.3561505.

Tiago Veiga received the MSc (2010) and Ph.D. (2015) degrees in Electrical and Computer Engineering from Instituto Superior Técnico, University of Lisbon, Portugal. He is a postdoctoral researcher at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU). Previously, he held a postdoctoral research position at the Institute for Systems and Robotics, Lisbon, Portugal, and an ERCIM Alain Bensoussan Research Fellowship at NTNU. His main research interests are in artificial intelligence, autonomous agents, planning under uncertainty, active perception, and adaptive behavior.

Hafiz Areeb Asad is currently pursuing a Ph.D. degree in information security and communications technology at the Norwegian University of Science and Technology, Trondheim, Norway. He received the M.Sc. degree in computer science from Uppsala University, Sweden, in 2020. He was a recipient of a Swedish Institute (SI) scholarship for global professionals. He did his B.Sc. degree in computer science from National University of Computer and Emerging Sciences, Islamabad, Pakistan in 2017. His current research interests include autonomous, cognitive and battery-less IoT.

Frank Alexander Kraemer received the Dipl.-Ing. degree in electrical engineering from the University of Stuttgart, Stuttgart, Germany, in 2003, the M.Sc. degree in information technology from the University of Stuttgart, and the Ph.D. degree in model-driven systems development from the Department of Telematics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2008. He is an Associate Professor with the Department of Information Security and Communication Technology, NTNU, and worked previously as a Technology Manager at a startup for IoT software that he co-founded. His current research interests include Internet-of-Things architectures and application development, embedded and autonomous sensor systems, and the application of statistical methods and machine learning in constrained settings.

Kerstin Bach is a professor in Artificial Intelligence at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU). Kerstin received her M.Sc. in Information Management and Technology (2007) and Dr. rer. nat. (Ph.D., 2012) from the University of Hildesheim, Germany. She worked as a research engineer at Verdande Technology (2013–2014) before joining NTNU. Her research interests are Artificial Intelligence methods for developing intelligence decision support systems involving both domain experts and end-users to create explainable, interpretable, and trustworthy AI systems. In particular, she works on data-driven and knowledge-intensive Case-Based Reasoning. She is the deputy head of the NTNU's Data and Artificial Intelligence group, program manager of the Norwegian Research Center for AI Innovation (NorwAI), and associated with the Norwegian Open AI Lab.