Future Generation Computer Systems 160 (2024) 92–108

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Equilibrium in the Computing Continuum through Active Inference

Boris Sedlak [*], Victor Casamayor Pujol, Praveen Kumar Donta, Schahram Dustdar

Distributed Systems Group, TU Wien, 1040 Vienna, Austria

ARTICLE INFO

Keywords:
Active Inference
Computing Continuum
Scalability
Edge intelligence
Transfer learning
Equilibrium

ABSTRACT

Computing Continuum (CC) systems are challenged to ensure the intricate requirements of each computational tier. Given the system's scale, the Service Level Objectives (SLOs), which are expressed as these requirements, must be disaggregated into smaller parts that can be decentralized. We present our framework for collaborative edge intelligence, enabling individual edge devices to (1) develop a causal understanding of how to enforce their SLOs and (2) transfer knowledge to speed up the onboarding of heterogeneous devices. Through collaboration, they (3) increase the scope of SLO fulfillment. We implemented the framework and evaluated a use case in which a CC system is responsible for ensuring Quality of Service (QoS) and Quality of Experience (QoE) during video streaming. Our results showed that edge devices required only ten training rounds to ensure four SLOs; furthermore, the underlying causal structures were also rationally explainable. The addition of new types of devices can be done a posteriori; the framework allowed them to reuse existing models, even though the device type had been unknown. Finally, rebalancing the load within a device cluster allowed individual edge devices to recover their SLO compliance after a network failure from 22% to 89%.

1. Introduction

Computing Continuum (CC) systems, as envisioned in [1–3], are large-scale distributed systems composed of multiple computational tiers. Each tier serves a unique purpose, e.g., providing latency-sensitive services (i.e., Edge), or an abundance of virtual, scalable resources (i.e., Cloud). However, the requirements that each tier must fulfill are equally diverse, as they span a wide variety of edge devices and fog nodes. Assume that requirements would be ensured in the cloud, e.g., by analyzing metrics and reconfiguring individual devices, massive amounts of data would have to be transferred. Also, if edge devices fail to provide their service to a satisfying degree, the latency for detecting and resolving this would be high.

Given the scale of the CC, requirements must be decentralized; this means that the logic to evaluate requirements must be transferred to the component that they concern. Cloud-level requirements, i.e., Service Level Objectives (SLOs), may thus be disaggregated into smaller parts that are ensured by the respective components. To contribute to high-level goals, each device optimizes its service according to its scope. This allows SLOs to span the entire CC, also called Deep SLOs [4]. While it is one challenge to segregate and disseminate SLOs, ensuring them is another. Requirements are versatile and may change over time, every component must itself discover how its SLOs are related to its actions. For this to happen, the device could use Machine Learning (ML) techniques to discover causal relations between its environment and SLO fulfillment [5]. This promotes the usage of Active Inference (AIF) [6], an emerging concept from neuroscience that describes how the brain continuously predicts and evaluates sensory information to model real-world processes. By extending individual CC components with AIF, they could develop a causal understanding of how to adjust their environment to ensure preferences (i.e., SLOs).

Ensuring SLOs autonomously (i.e., evaluating the environment to infer adaptations) makes components intelligent [7]; any system composed entirely of such intelligent, self-contained components becomes more resilient and reliable. No central logic must be employed to ensure SLOs; thus, higher-level components can rely on the SLO fulfillment of underlying components. Ascending from intelligent edge devices, the next level would be intelligent fog nodes; those we see in the ideal position to orchestrate the service of edge devices. Thereby, edge devices in proximity are bundled into a device cluster, administered by a fog node; whenever the Edge is scaled up with new devices (or device types), existing SLO-compliance models can be exchanged within the cluster. While each tier has its own SLOs, their tools for adaptation can have a different scale, e.g., fog nodes would be able to shift computations within clusters from devices that fail their SLOs. Such operations can consider environmental impacts (e.g., network issues) as well as heterogeneous device characteristics. The Cloud, as the next layer, would even have sweeping tools to ensure global SLOs.

* Corresponding author.
E-mail addresses: boris.sedlak@dsg.tuwien.ac.at (B. Sedlak), v.casamayor@dsg.tuwien.ac.at (V.C. Pujol), pdonta@dsg.tuwien.ac.at (P.K. Donta), dustdar@dsg.tuwien.ac.at (S. Dustdar).

https://doi.org/10.1016/j.future.2024.05.056
Received 28 November 2023; Received in revised form 23 May 2024; Accepted 28 May 2024
Available online 30 May 2024

To realize this vision, we present our framework for collaborative edge intelligence. Guided by AIF, individual edge devices gradually develop a causal understanding of how to ensure their SLO. This knowledge is federated through a device cluster; edge devices of arbitrary types reuse existing models to ensure their SLOs. Thus, the entire Edge becomes spanned with SLO-compliant devices, which allows other CC tiers (i.e., up to the Cloud) to construct their service on top of that. By the same method, cluster leaders infer how to adjust their environment; thus, each tier may achieve an equilibrium for the compound service offered. Hence, the contributions of this paper are:

- An AIF-based ML technique that allows CC components to gradually identify causal relations between environmental metrics and SLO fulfillment. Components can thus evaluate SLOs decentralized and update their beliefs according to new observations.
- The transfer and combination of ML models between heterogeneous devices to accelerate their convergence towards SLO-fulfilling configurations. This simplifies the onboarding of new device types (i.e., horizontal scaling) on the Edge.
- An offloading mechanism that redistributes load in an edge–fog cluster according to devices' capabilities to fulfill high-level SLOs. Thus, it counters environmental factors and improves the cluster-wide level of QoS and QoE.

The remainder of this paper is organized as follows: Section 2 introduces background knowledge and related work as a prerequisite for presented concepts. Section 3 presents our framework for collaborative edge intelligence. Section 4 contains the prototypical implementation of the framework and the evaluation methodology; the respective results are presented in Section 5. Finally, we summarize our paper in Section 6.

2. Preliminaries

2.1. Background

The framework presented in this paper builds heavily on two existing concepts that we adapt for our usage, namely causality and AIF. Although these topics might be known to some readers, we provide this section to ensure a solid understanding of their core aspects and terminology. Furthermore, since both concepts are not native to computer science (or distributed systems), we highlight existing intersections as far as possible.

2.1.1. Causality and causal network graphs

Causality allows modeling causal relations between events or variables. While spurious correlations are misleading and hide the true causes, causality answers *why* an event happened. However, to identify causal relations, specific experiments and consideration of expert knowledge are required. To define a general theory of causality, Pearl [8] proposed Structural Causal Models (SCMs). Such a mathematical model can be expressed through causal graphs, e.g., as Directed Acyclic Graph (DAG). Thus, variables can be arranged from cause to consequence.

Causality is a hot topic in research because of its ability to provide explanations for phenomena through interpretable graphical models. This is why many works link causality and machine learning; see [9] for a comprehensive review. Thereby, causality can also be embedded into distributed systems, e.g., for root cause detection [10]. As another instance, *Lin et al.* [11] use causal graphs in Cloud computing to detect dependencies within a microservices-based architecture. For such use cases, DAGs are an ideal modeling tool. Interestingly, they monitor SLOs to trigger causal inference over their causal graphs, being able to detect the source of the SLO violation.

Another crucial concept for our work – or generally for scalability in the CC – is the Markov Blanket (MB). Consider a Bayesian network (BN) represented as a DAG (e.g., Fig. 3): a random variable is conditionally independent of all other variables, given its MB. In other words, the MB of a variable *shields* it from external variables. In a DAG, the MB of a variable consists of its parents, children, and co-parents. Discovering the structure of BNs and extracting MBs through data is not a simple task, and many works are devoted to that; see [12] or [13] for specific techniques, and [14] for a thorough survey. Regardless of the system size, MBs can achieve modularity; thus, the system can be managed and controlled on a convenient scale.

Graph-based causal models promise to make systems explainable. Inspired by that, our work stems from [2,15] to build MBs around SLO-governed components. Thus, it becomes possible to isolate the system variables that affect SLO fulfillment. On the one hand, this drastically reduces the number of variables required for analysis thanks to conditional independence; the system can thus be managed at scale. On the other hand, it is possible to leverage the BN to explain causal effects between variables in the MB and the SLOs' behavior (e.g., failure).

2.1.2. Active inference

In this work, we use AIF to extend devices with causal knowledge on how to fulfill their SLOs. However, we consider AIF an unknown concept for most readers outside of neuroscience; therefore, we use this section to summarize core concepts of AIF according to *Friston et al.* [16–21].

Core concepts. To interpret observable processes, agents generate models that resemble these processes, e.g., humans reason that it rains due to water drops falling from the sky. However, if this generative model and the real-world process diverge, the agent will eventually be "surprised", e.g., because water drops were actually caused by a neighbor watering her plants. The discrepancy (or uncertainty) between the agent's understanding of the process and the reality is called Free Energy (FE). In simple terms: the lower the FE, the higher the prediction accuracy.

Internally, agents organize generative models in hierarchical structures; each level interprets lower-level causes and, based on that, provides predictions to higher levels. For example, suppose (1) it rains with a certain probability, (2) I bring an umbrella. This is commonly known as Bayesian inference and allows agents to use priors (i.e., existing beliefs) to calculate the probability of related events. Thus, decision processes can be segregated into self-contained causal structures (i.e., MBs) that share only a limited number of interface variables. For example, only the weather state (*rainy* or *sunny*) is considered for picking the umbrella; any lower-level observations that determined the agent's perception of the weather (e.g., humidity or illumination) are disregarded.

To decrease FE, AIF agents repeatedly engage in action-perception cycles by (1) predicting outcomes, (2) awaiting (or seeking) the outcome, and (3) updating beliefs. This phase is known as predictive coding. Afterward, they can actively adjust the environment to their beliefs. As generative models become more accurate, causal relations between their preferences (e.g., SLOs) and the environment are revealed. However, the ability of agents to discover causal relationships is highly dependent on the number and accuracy of observations [22]. Fortunately, the CC provides large amounts of operational metrics.

Some aspects of AIF, in particular decision-making, intersect with reinforcement learning. Notably, the two approaches are not mutually exclusive, on the contrary, they are complementary as shown by existing works [18,23,24]. Important differences of AIF are that agents are biased when they try to adapt the exterior towards their beliefs and that they are specialized in minimizing surprise for an empirically verifiable model.

Intersection with distributed systems. Considering presented works, most research on AIF has not been embedded and evaluated in operative distributed systems (e.g., [19,25]). To the best of our knowledge, our latest research [26] is thus among the few works that embedded AIF into distributed systems; another work that we want to highlight is *Levchuk et al.* [27], which created a decentralized mechanism for team adaptation. For the remaining paper, our work in [26] serves as a reference on how AIF agents can infer SLO-compliant device configurations: agents operate parallel to continuous processing and adapt their generative models according to prediction errors. We call such a model – at its core a BN – an Equilibrium-Oriented SLO-Compliance (EOSC) model. In this paper, we will extend the EOSC model to achieve equilibrium in the CC.

2.2. Related work

This section provides recently published related works that discuss (1) the training and application of causal ML models on the Edge, (2) transfer learning approaches in the CC, and (3) methods of load balancing and computation offloading that are popular across the CC. Following that, we highlight for each of these fields the research gap that our work aims to fill.

2.2.1. Causal ML training on the edge

Sudharsan et al. [28] developed an `Edge2Train` model to analyze real-time data on the fly. With Edge2Train, Support Vector Machine (SVM) models are trained offline at edge nodes using real-time IoT. *Chen et al.* [10] use causal inference (`CauseInfer`) mechanisms to pinpoint the root causes within the system; for this, CauseInfer explicitly determines fault propagation paths. A similar approach (called `Nazar`) is designed by *Hao et al.* in [29], which applies mobile devices to find root causes in distributed systems. Through experiments, Nazar confirmed that models can be improved due to cause-specific adaptation while monitoring large numbers of devices. Zhang et al. presented *Octopus* [30], which ensures SLO fulfillment during a CV task; for this, they used deep RL to provide the optimal device configuration for three parameters.

There evidently exists work that identifies and applies causal understanding to ensure system requirements; however, with the exception of Nazar [29], they treat model training as a one-time process. Hence, drifts (or shifts) in the variable distribution stay undetected. Further, it is impractical to assume that initial training data suffices to create causal understanding; this is also a shortcoming of Nazar. Contrarily, our approach uses AIF to gradually create causal models over multiple iterations and continuously ensures model accuracy by updating beliefs according to prediction errors. Unlike presented work (e.g., *Octopus*), AIF intrinsically seeks to ensure both an accurate device model, as well as fulfill agents' objectives (e.g., high SLO fulfillment). Although RL could rebuild this behavior through multiple values functions, this presents a challenge in itself, which can serve as a comparison for the results of this paper.

2.2.2. Transfer learning in the CC

Goyal et al. present `MyML` [31], a hardware-friendly model transfer for edge nodes. MyML uses transfer learning to create small, lightweight, custom ML models based on user preferences. *Wu et al.* present a novel approach to online transfer learning for both heterogeneous and homogeneous labels of multi-source domains [32]. This approach is very efficient in online classification, and the weights are dynamically adjusted depending on the source domain. *Hsu et al.* provide a clustering mechanism that considers the similarity of domains and tasks for transfer learning [33]. They provided a function based on domain similarities used for cross-task transfer learning.

Transferring ML models is an important measure for relieving resource-restricted devices from training; this can consider recipients' context to provide a tailored model. However, the presented works did not consider low-level hardware characteristics to identify potential teachers among nearby devices. To that extent, our framework uses hardware classification to find adequate models within a device cluster and creates a tailored model by merging conditional probabilities of BNs.

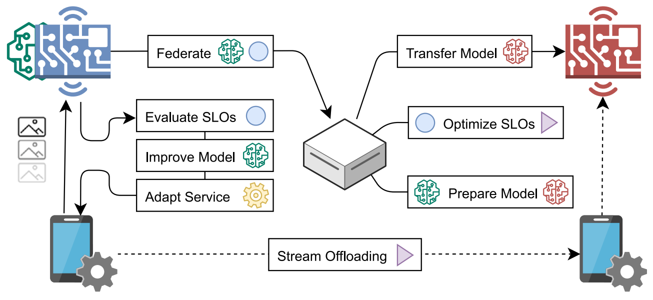


Fig. 1. High-level overview of the collaborative edge intelligence framework that continuously improves model evidence, shares this knowledge between edge devices, and optimizes SLO fulfillment within this cluster.

2.2.3. SLO-induced load balancing and offloading

Elasticity is one of the most effective ways to ensure the requirements of dynamic workloads by automatically provisioning or deprovisioning resources based on demand [34]. `SLOC` is a novel elastic framework developed by *Nastic et al.* in [35], that allows users to provide and consume cloud resources in an SLO-native manner while guaranteeing performance. Further, *Furst et al.* bring elastic service principles from the cloud to edge computing [36]. They evaluated elastic and non-elastic services at the edge while processing images to latency SLOs, and noticed improved service provisioning through elasticity. In [37], *Menino* proposed efficient failure detection mechanisms for unstructured overlay networks. This approach aims to identify efficient neighborhood overlays, which dynamically identify and maintain each node in P2P networks.

SLOs are an efficient way for modeling and enforcing requirements at the respective component. Nevertheless, the remaining question is whether components have the required scope to recover SLO failures (e.g., by offloading computation), but it is impractical to evaluate SLOs in the cloud (e.g., MHP2P). Ad-hoc hierarchical structures could provide a remedy, which *Menino* [37] are the only ones to use among the related work. However, they all assume prior knowledge of which variables impact SLO fulfillment. Contrarily, our approach (1) gradually increases the SLO scope by forming device clusters that span the entire CC, and (2) evaluates causal relations among environmental variables to shift the load from impacted devices.

3. Collaborative edge intelligence

To ensure SLOs throughout computational tiers, we propose our framework for collaborative edge intelligence that encompasses three main contributions: (1) The continuous model optimization based on AIF, which ensures SLOs (locally) on a device basis; (2) the federation and combination of EOSC model between edge devices, which decreases the overhead of training models for different device types from scratch; and (3) the evaluation of SLOs on a cluster-level, which can rebalance load within the cluster according to environmental factors.

These three contributions are described in the respective Section 3.1 to 3.3; Fig. 1 contains a high-level overview of the framework's capabilities. On the left, it is depicted how SLOs are evaluated to continuously train an ML model and adapt the service accordingly; this model is then federated and combined at a fog node, which provides the model to an unknown device type (marked as red). The fog node analyzes the overall SLO fulfillment in the cluster; if it appears beneficial to offload computation from one device to another one (e.g., from the blue to

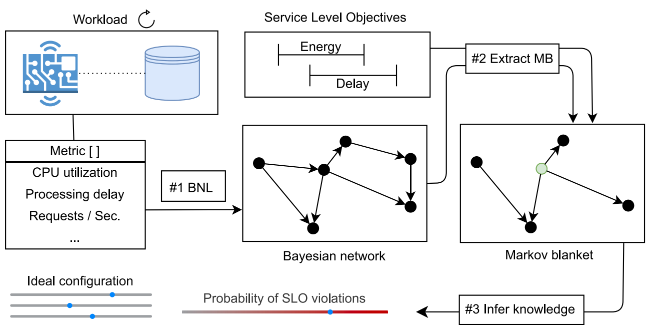


Fig. 2. Training a Bayesian Network from processing metrics (#1); this is used to extract the minimum number of variables related to SLO fulfillment (#2) and a configuration that satisfies them (#3).

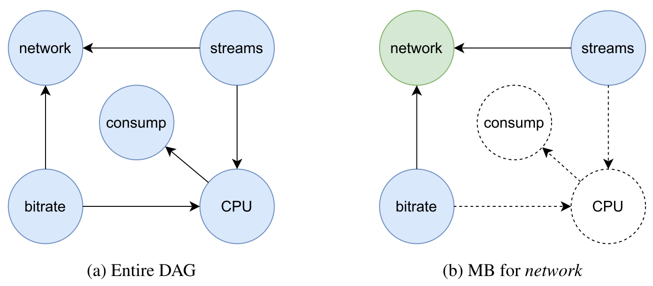


Fig. 3. Causal variable relations in the DAG of a trained BN.

the red one), this is orchestrated by the fog node. Logically, the model transfer and load balancing rely on the SLO fulfillment in the Edge; this is why all three contributions are required to ensure SLOs on multiple tiers (or the entire CC).

3.1. Continuous model optimization

An accurate generative model allows to explain a system's behavior (e.g., why SLOs were violated), infer how to adapt the system to ensure SLOs, and predict how changes will affect this. Further, prediction errors are propagated back to the agent so that the model can be improved according to the experienced deviations. In the following, we will first present the representation of the EOSC model and the applied training method. Afterward, this process is integrated into an AIF agent, which uses this process to continuously improve the model accuracy.

3.1.1. Static model training and inference

Within previous work [5], we presented the idea of obtaining a generative model from processing metrics and inferring system configurations that fulfill SLOs. However, it lacked a formal implementation; this will be the content of this section. Fig. 2 summarizes our method to train the BN, which is required as a causal structure for our framework:

To report their current state, edge devices produce metrics throughout processing; this data can be used to create a generative model through Bayesian Network Learning (BNL) (#1). This reveals (ideally) causal dependencies between variables, including the impact of environmental changes (e.g., increased incoming requests). To decrease the model complexity, we identify the minimum number of variables relevant to fulfill system requirements (i.e., SLOs); this subset is the MB of the BN (#2). Given the MB, we estimate the probability of SLO violations for different hypothetical scenarios and (#3) infer the device configuration with the highest statistical compliance level. In the following, we elaborate on these substeps further.

Bayesian network learning. BNL is an efficient way to generate the most accurate structure from given data; its two main parts are STRL – structural learning of causal dependencies (i.e., DAG), and PARL – parameter learning as quantification of variable dependencies.

While there exist numerous BNL techniques [38], we focus on Hill-Climb Search (HCS) for structure learning and Maximum Likelihood Estimation (MLE) for parameter learning due to their rapid convergence, low complexity, and efficiency when considering limited attributes. For a data set with 5 columns, the resulting DAG could look like Fig. 3(a). The AIF agent uses these methods for constructing (and later updating) the EOSC model: STRL trains a DAG through HCS; PARL evaluates the conditional variable dependencies through MLE. Together, they can be used to create a BN $model$ from data D as $model = \mathtt{PARL}(\mathtt{STRL}(D), D)$.

Markov blanket selection. A BN contains by design directed relations and conditional dependencies of random variables; however, to determine the state of an individual node x, only a share of the BN nodes are influential. This promotes the application of MB [2,39,40], which shield a variable from all nodes that are conditionally independent of it. Suppose we specify an SLO according to device capabilities (e.g., network throughput $< t$) and evaluate it using a single variable (e.g., *network*), we want to identify metrics related to SLO fulfillment. Namely, these are all variables contained in the MB of *network*; the function $\mathtt{MB}(model, network)$ thus returns all blue nodes in Fig. 3(b).

In this context, we distinguish between metrics that statically reflect the system state (e.g., *CPU*), and those that represent a parameterizable variable (e.g., *bitrate*). However, we summarize both using the term "metrics" from a BNL perspective. While static metrics are essential to explain why an SLO is in its current state, only parameterizable ones can be dynamically reconfigured, i.e., they are the possible action states of the AIF agent. Overall, the sum of metrics in the MB provides a clear understanding of why an SLO is in its current state.

Knowledge extraction. There exist two main categories of algorithms for extracting knowledge from BNs, namely Approximate Inference (AxI) and Exact Inference (EI). Given a BN and system requirements (i.e., SLOs), we seek to extract probabilities of SLO violations under different environmental states. This mechanism works equally for different CC tiers; an edge device, for example, could use its BN to answer $P(network > t)$, with t being a custom threshold. For dynamic reconfiguration, we require inference to be (1) accurate, (2) converge reliably, and (3) fast for large networks. We argue that EI and, in particular, Variable Elimination (VE) [41] fulfill these constraints:

Consider the DAGs from Fig. 3: We construct a QoS SLO that is fulfilled if *network* is below t and infer the probability of SLO violations for different variable assignments. VE accepts a list of target variables (T), variable assignments (A), and an elimination order (O). VE iterates over O and eliminates model variables while updating the beliefs of remaining nodes; the graph thus eventually contains only T. To decrease the complexity of VE, we execute it on $mb = \mathtt{MB}(model, network)$, the node list thus equals $\{network, streams, bitrate\}$. Later, we call VE through $\mathtt{INFERENCE}(m_x, T, A)$, where m_x can be any subset of the BN. If we execute INFERENCE with *mb*, $T = \{network\}$, $A = [(streams : 2), (bitrate : 720)]$, and arbitrary O, the result contains all conditional probabilities of $network$ given the variable assignment; from this we can extract $P(network > t)$.

This is our central mechanism for identifying probabilities of SLO violations given a system state. If an SLO is violated due to an environmental change, e.g., higher $streams$ and thus $network$ exceeds t, we can compare possible configurations and provide the one with the highest probability of fulfilling the SLO. In the given example, only *bitrate* can be parameterized (i.e., configured); to fulfill the $network$ SLO, the corresponding measure could thus be to decrease $bitrate$. This matches our envisioned level of intelligence, i.e., "understanding a situation and reacting according to needs", and neatly fits the principles of elastic computing [34].

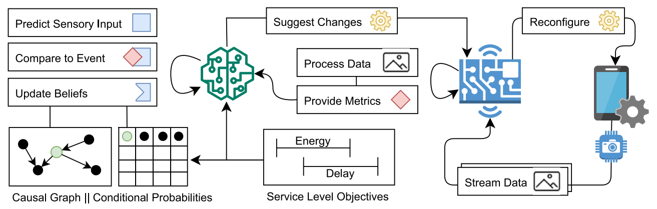


Fig. 4. Overview of the active inference cycle — learning how to fulfill SLOs by adapting the generative process.

3.1.2. Active inference cycle

The tools presented in the last section created a BN from processing metrics, extracted an MB, and inferred system configurations that fulfill given SLOs. Supposed there is sufficient data available, BNL can be a one-time process; however, there are two fundamental issues: (1) data shifts, which likely occur after some time, will inevitably distort the accuracy of the ML model, and (2) it is impractical to empirically evaluate how an exponential number of system configuration impacts SLO fulfillment. Large and complex systems, such as the CC, require a different approach: creating and updating a model incrementally according to new observations while drawing conclusions for unknown parameter combinations from existing data.

To evaluate this parameter space of configurations, we extend the AIF agents from [5] to interpolate between empirically evaluated combinations; to maintain the model's FE low, agents continuously update conditional probabilities of variable relation according to new observations. By design, our AIF agents can be employed at any CC tier; nevertheless, the running example in this paper is focused on intelligent edge devices, which collaborate under the supervision of a fog node. Thus, we raise the granularity of intelligence from the Edge to the Fog. In the following, we present the different tasks executed by an AIF agent; this includes training and updating the BN, as well as evaluating its scope of actions according to a set of behavioral factors. Based on that, agents decide how to modify the system; each of these changes is again reflected by system metrics.

Agent and operation. The AIF agent operates parallel to regular device tasks, e.g., serving clients. Although regular operation, model training, and inference are logically separated, they take place on the same physical device; Fig. 4 contains a visual representation: assume an edge device that continuously performs a workload, e.g., processing client data. The agent observes the device state and the environment through metrics; thus, it can evaluate whether processing complies with SLOs, e.g., if a request was finished with $delay < t$. From that data, the agent creates a BN, where conditional probabilities reflect the SLO fulfillment under a discrete environmental state. Then, the agent starts with predictive coding, i.e., forecasting whether future events will fulfill SLOs, comparing the expectation with actual observations, and updating the BN accordingly.

After each iteration, the agent infers how to modify the system configuration to optimize local SLO fulfillment. Following that approach, the AIF agent can create a generative model from scratch or update a BN according to new observations by following its sensing-acting loop. Thus, it is possible to cancel out data shifts, e.g., the result of a model transfer from one edge device type to another. AIF can therefore perform the fine-tuning that is required after such an operation.

Free energy minimization. To create an accurate model, the AIF agent operates in cycles; each cycle processes a $batch$ of observations that reflects the environmental state, including the latest system configuration. The agent continuously evaluates the $batch$, updates its $model$, and chooses which system configuration (c_{next}) to choose for the next iteration. Throughout these cycles, the AIF agent has one central goal: decreasing the FE, or in other words, minimizing surprise of predictions. Therefore, we will first present how we calculate surprise and then embed it into the high-level loop executed by the agent.

Algorithm 1 SURPRISE for model and batch

```
Require: $model$, $batch$, $V_{SLO}$
Ensure: $\mathfrak{J}$ // surprise over all observations
 1: $\mathfrak{J} \leftarrow 0$
 2: $mb \leftarrow \mathtt{MB}(model, V_{SLO})$
 3: for each $var$ in $V_{SLO}$ do
 4:     $log\_likelihood \leftarrow 0$
 5:     $ev \leftarrow \mathtt{MB}(model, var)$
 6:     for each $row$ in $batch$ do
 7:         $evidence \leftarrow row \cap ev$
 8:         $p \leftarrow \mathtt{INFERENCE}(mb, var, evidence)$
 9:         $log\_likelihood \leftarrow log\_likelihood + \log(p)$
10:     end for
11:     $cpt \leftarrow \mathtt{CPT}(model, var)$
12:     $k \leftarrow |cpt|$ // number of states in the CPT
13:     $n \leftarrow |batch|$
14:     $bic \leftarrow (-2) \times log\_likelihood + k \times \log(n)$
15:     $\mathfrak{J} \leftarrow \mathfrak{J} + bic$
16: end for
17: return $\mathfrak{J}$
```

For calculating the surprise for $batch$ and $model$ we present Algorithm 1. To decrease the complexity, we limit the calculation to variables that directly reflect SLO fulfillment (V_{SLO}), and execute INFERENCE only on the MB of V_{SLO} (Line 2). This node set is further filtered (Line 5) to contain only the evidence variables (ev) that impact the outcome of var; afterward, in Line 7, each row in the $batch$ is filtered to contain only these variables. In Lines 8 & 9, the probability of observing var, i.e., the state of the SLO, given the environment ($evidence$) is first inferred and then appended as $log\_likelihood$. For each var, the cpt from $model$ is considered, from which k – the number of states – can be extracted as a representation of model complexity. CPT is as a helper function to get the CPT for a var in $model$. Together with n – the number of observations – the BIC is calculated (Line 14). After calculating the surprise for each $var \times row$, this overall sum is returned.

The surprise has a special role within our AIF cycle, as it determines when and how BNL takes place; consider therefore Algorithm 2, which shows the high-level loop executed by the AIF agent. At the beginning of each iteration, the agent ensures that there exists a model, otherwise, it creates an initial structure from $batch$ (Lines 1 & 2). Notice, that STRL and PARL accept now another parameter – $model$ – which allows to update the DAG and CPTs of $model$ according to $batch$. Whether STRL or PARL is executed (Lines 7–11) is determined by the surprise magnitude (s). If s exceeds the median surprise of the last 10 rounds (m_{10}) by a custom factor h, STRL is applied; otherwise, if s exceeds m_{10}, PARL is applied. This distinction is necessary because STRL and PARL have quite different runtimes, as we will reveal in Section 5. Finally, in Lines 12 & 13, the agent evaluates possible system configurations and determines which one it will use for the following iteration. We will explain these two functions in the next two paragraphs.

Behavioral factors. The behavior of the AIF agent, i.e., how it selects between possible system configurations, is determined by three major factors: The pragmatic value (pv) defines how well the device fulfilled client expectations, e.g., if a streamed video's resolution is satisfactory. The risk assigned (ra) determines how likely the system will fail its service, e.g., if the stream packets are delivered on time. Lastly, the information gain (ig) represents the agent's expectation of how much it can improve model accuracy. The ig is directly related to surprise minimization, whereas pv and ra reflect the agent's capability to fulfill SLOs. To separate concerns, we divide SLOs according to their characteristics: pv represents QoE requirements, while ra contains QoS requirements. Combined, these three factors determine the behavior of the agent; in the following, we will calculate each of them.

Algorithm 2 An Iteration in the AIF Cycle

```
Require: $model$, $batch$, $\mathfrak{I}$, $h$, $V_{SLO}$
Ensure: $c_{\text{next}}$ // Next configuration
 1: if $model = \emptyset$ then
 2:     $model \leftarrow$ PARL(STRL($\emptyset, batch$), $batch$)
 3: end if
 4: $s \leftarrow$ SURPRISE($model, batch, V_{SLO}$)
 5: $\mathfrak{I} \leftarrow \mathfrak{I} \cup \{s\}$
 6: $m_{10} \leftarrow median(\mathfrak{I}_{10})$ // over the last 10 values
 7: if $s > (m_{10} \times h)$ then
 8:     $model \leftarrow$ STRL($model, batch$)
 9: else if $s > m_{10}$ then
10:     $model \leftarrow$ PARL($model, batch$)
11: end if
12: $K \leftarrow$ CALCULATE_FACTORS($model$)
13: $c_{\text{next}} \leftarrow$ BEST_CONFIGURATION($K$)
14: return $c_{\text{next}}$
```

To infer the optimal device configuration (i.e., the one with the highest SLO fulfillment), the agent limits itself to finding the Bayes-optimal configuration [42], i.e., the optimal under current knowledge. Therefore, the AIF agent first infers the assignment for known parameter combinations (c_k) that were empirically evaluated and then interpolates between these values to span the entire parameter space. Calculating pv and ra is similar to Algorithm 1 (Lines 5–8): It requires a subset $V_Q \subseteq V_{SLO}$ – either QoS or QoE SLOs – which is used as $ev \leftarrow \mathrm{MB}(model, V_Q)$. For each row in c_k, $evidence$ is constructed equally, so that $\mathrm{INFERENCE}(mb, V_Q, evidence)$ provides the joint probability of all QoS or QoE violations.

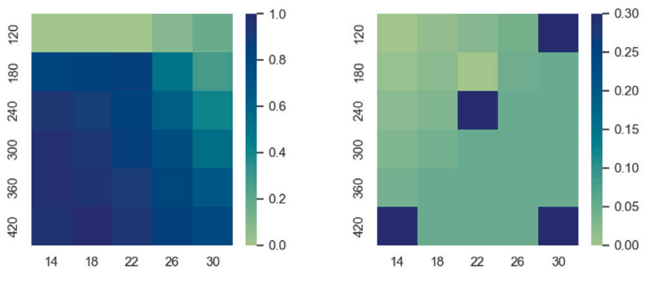
$$ig(c) = e + \left(\frac{\tilde{\mathfrak{I}}_c}{\bar{\mathfrak{I}}}\right) \times 100 \tag{1}$$

In accordance with [26], high surprise indicates high information insight and, hence, possible improvement of the model precision. However, from an agent's perspective, is it worth abandoning a supposedly satisfactory configuration (in terms of pv and ra) to search for a global optimal one? This presents a tradeoff between exploration of unknown areas and the tendency to stick to exploited areas; multi-agent systems commonly model this through hyperparameters (e.g., [27]). In our case, we calculate the ig of a configuration $c \in c_k$ as presented in Eq. (1) [26]: it compares the median surprise ($\tilde{\mathfrak{I}}_c$) for c with the overall mean surprise ($\bar{\mathfrak{I}}$). Configurations with high $\tilde{\mathfrak{I}}_c$ will thus be preferred by the AIF agent.

Parameter space. The AIF agent calculates the behavioral factors for all entries in c_k and summarizes them as K (Line 12 of Algorithm 2). For the next step, imagine two configuration parameters $\{fps, pixel\}$ with their combinations arranged in a 2D $[fps \times pixel]$ matrix. After calculating K, the blank spaces in the parameter matrix are filled by performing linear interpolation[1]. As a potential result, consider the matrix depicted in Fig. 5(a). Later, in Section 4.2, the agent will interpolate in a 3D parameter space.

Contrarily to pv and ra, the agent does not apply interpolation to estimate the ig of an unknown parameter configuration. Instead, in the absence of observations for c, it assumes that $ig(c) = \max(\mathfrak{I})$. Further, it remains to introduce a hyperparameter from Eq. (1), namely e. To improve the interpolation of pv and ra, the agent initially focuses on key positions of the possible configurations. Fig. 5(b) illustrates that tendency; the visually highlighted blocks are increased by $e = 0.3$. When calculating the behavioral factors, the AIF agent thus initially focuses on these cornerstones to set up the interpolation; after visiting c, it subtracts e from $ig(c)$.

[1] In fact, this is done using Python scypy, which triangulates data through a convex hull to perform linear barycentric interpolation on each triangle.



(a) Interpolation for pv (b) ig after 1 round

Fig. 5. Matrices of behavioral factors used by the AIF agent.

To summarize possible risks but also benefits that emerge from a configuration c, we combine the three factors under a common one (u) that we calculate as $u_c = pv_c + ra_c + ig_c$. The AIF agent compares common factors of all possible configurations and selects the highest-scoring (Line 13 of Algorithm 2). By repeating this cycle, the agent gradually develops an understanding of which areas in the parameter space are more likely to fulfill SLOs, e.g., the left-bottom area in Fig. 5(a).

This concludes the agent's continuous model optimization, which maintains an up-to-date model of a processing task (i.e., the generative process). The high accuracy in the EOSC model allows the AIF agent to infer (Bayes-)optimal device configurations, which ensures QoS and QoE of ongoing operation. In the following, we will now focus on the collaboration between the Edge-based agents.

3.2. Knowledge transfer within the cluster

By now, we presented AIF agents that can create generative models from scratch or update a model according to new observations. However, if we assume a cluster of nearby devices that process similar workloads, training EOSC models for every device seems redundant. Also, if we aim to extend the cluster with more devices (i.e., scaling up horizontally), model training delays the time until devices operate according to requirements. Instead, we envision the federation of knowledge between edge devices by exchanging EOSC models within the device cluster. Such a transfer learning approach appears to be a straightforward process if the models were trained in the exact same environment [32]. However, the Edge is composed of multiple heterogeneous device types; the resulting models thus reflect the characteristics of the device it was trained on, i.e., its capability to cope with SLOs depends on the processing hardware. For example, a multi-core device is certainly capable of processing multiple video streams, while a single-core one is not. Furthermore, the behavior of AIF agents (i.e., which action it takes) and environmental dynamics (e.g., demand) determine which parameter combinations get more or less exploited.

Whenever a new device (type) joins a cluster, the question is whether there exists a device within the cluster whose environment and characteristics match the newly-joint device's. Meanwhile, devices present in the cluster share their EOSC models and device characteristics (e.g., hardware specs or environmental factors) with the cluster leader (i.e., standing hierarchically above the device cluster). As a new device joins the cluster, its characteristics are compared with the present ones to select a fitting model. In cases where the characteristics of multiple devices are similar, their models are merged and provided to the newly-joint device. Thus, the newly-joint device builds its EOSC model on top of existing knowledge in the federation.

In the following, we dive deeper into this transfer-learning process by answering (1) how models are federated between devices, (2) how hardware characteristics are compared to select a model, and (3) how models are combined to fit a target device.

3.2.1. Cluster-wide model exchange

The EOSC model exchange knows two roles: (1) consumer — when joining a device cluster it might be preferable to adopt an existing model rather than training one, and (2) provider — any device might itself share its model with devices that join the cluster. The selection of a fitting model, however, can happen on any trusted device; we assume for this task either a cluster leader (i.e., an outstanding device elected due to its capabilities) or a powerful fog node. To provide an estimation, these models are supposedly smaller than 2 MB, as measured in [5,26].

When making the architectural decision (i.e., cluster leader or fog node), various factors can be considered, among them: network scale, cost, geographic location, and availability. In cases where the cluster would be small (e.g., 10 devices), an edge device (e.g., from Table 3) could cope with collecting and preparing EOSC models; however, for larger clusters (e.g., 1000 devices), regular edge devices might fail to do so. In any case, a strong factor for using fog nodes is their high availability — fog nodes can reliably cache a high number of models from various devices. Either choice, they assume equal responsibilities, thus we call them simply *leader node*. This leader node periodically collects EOSC models of devices registered in the cluster, as well as their hardware characteristics. Based on this information, models will be provided for new device types.

3.2.2. Model comparison and selection

Transferring a EOSC model to a newly-joint device raises two questions: First, is the transfer of an existing model more efficient than learning the model from scratch? And second, how to choose the most convenient model for the new device? Of course, the second question assumes that the device type is unknown and the cluster does not contain the respective trained models so far. The first question will be answered and discussed as a result of this article; the second question, however, requires building a hypothesis around how to choose a model.

The dynamism within the training environment has a decisive impact on the resulting model: applications with a stable number of user requests do not suffer many dynamics, while applications that are linked to specific events (i.e., disaster management) can experience extremely different requirements. However, we assume that environmental factors are out of our hands — we are unaware of the dynamics of the environment in which the device is set. Due to that, we focus on the device characteristics when transferring models between edge devices. To that extent, we get inspiration from the work of *Casamayor et al.* [43], which allows classification of heterogeneous characteristics of the devices found in a cluster, namely their CPU and GPU capacity. This means that we relatively classify the CPU capacity (p) of the devices in the cluster in a range $[p_{min}, p_{max}]$, and their GPU capacity (g) from $[g_{min}, g_{max}]$. Given that there are numerous edge devices without GPU, it is possible to set $g_{min} = 0$. To make this more tangible, in Section 4.2, we present a list of edge devices whose hardware is classified accordingly. Finally, we define each device's capacities as $dc = p + g$. To estimate the similarity of device characteristics and to identify a device with a matching model, the leader node selects the device(s) with the closest integer dc.

3.2.3. Combination and preparation of models

Heterogeneous edge devices differ in terms of hardware characteristics. Using the presented mechanism, there would frequently occur situations in which there is not exactly one device that trumps all others. For example, consider a device with type t_x that joins a device cluster; there are already numerous device types present, among them t_a and t_b. The leader node classifies their capabilities as $dc_a = 3, dc_b = 5$, and $dc_x = 4$. Which model should now be provided to t_x, the one trained on t_a or on t_b? And in case $dc_a = 2$ and $dc_b = 7$; is choosing dc_a really the smartest choice?

What we envision for both cases is merging the models from t_a and t_b, thus creating a new model m_{ab} that presents the intersection. In the second case, where dc_x does not fall exactly between dc_a and dc_b, this is done proportionately. Therefore, we require a mechanism to combine EOSC models — still BNs at their cores. To date, merging BNs is an ongoing research field that still presents various limitations [44,45]; in most cases, it is coupled to conditions that models must fulfill. Due to this, we limit our work to merging CPTs. As long as two models m_a and m_b contain the same structure (i.e., their DAGs are identical) and their CPTs have the same cardinality (i.e., variable states), this is done as follows: For a random variable r and its $\mathtt{CPT}(m, r)$, each table cell's expected value (P) is calculated as shown in Eq. (2); P_a and P_b represent probabilities of m_a and m_b, the coefficients w_a and w_b reflect the distribution of dc_x between dc_a and dc_b. For example, if dc_x is aligned centrally between them, they take the value $w_a = w_b = 0.5$; otherwise, it is shifted proportionally, but $w_a + w_b = 1$ must remain true.

$$P_x = (w_a \times P_a) + (w_b \times P_b) \tag{2}$$

If m_a and m_b do not fulfill these requirements, they would have to undergo a transformation process. Nevertheless, in Section 4.2.2, we apply a workaround to merge BNs whose CPTs have different cardinalities. After merging the EOSC models, the leader node provides m_{ab} to the newly-joint device; once received, transfer learning is completed. Thus, it decreases the time for model training or even skips it entirely.

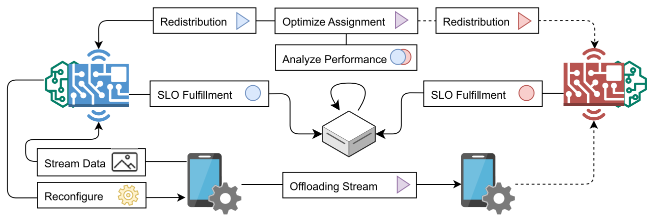


Fig. 6. Evaluating SLOs within a device cluster and reassigning tasks.

3.3. Stream offloading in the edge–fog cluster

Regardless of whether trained by an AIF agent or transferred from another device, a EOSC model is a decisive step towards SLO fulfillment. Thus, edge devices are continuously reconfigured to achieve maximum SLO compliance. However, despite our efforts, edge devices are still vulnerable to environmental factors that cannot be controlled, e.g., irregular peaks in client traffic. While a EOSC model can have a hard time finding an SLO-compromising device configuration, idle edge devices in close proximity might be available for offloading computation. Again, to match our desired level of intelligence, this can be achieved through collaboration between the agents. Given that the struggling edge device is part of a device cluster, it is possible to (1) compare the device's capabilities to fulfill their SLOs within their environment, and (2) balance the load accordingly. Notice, that shifting the load within the cluster is a (local) reconfiguration that follows the same rules as in Section 3.1; this time, however, on a higher level.

In the following, we describe how to evaluate, analyze, and optimize the cluster-wide SLO compliance; the overall process is visible in Fig. 6: The edge devices in the cluster (red & blue) serve their respective clients, e.g., by processing data, which is subject to dynamic reconfiguration according to the EOSC model. Throughout processing, the edge devices supply their SLO fulfillment to the leader node. Among that, they provide other factors (i.e., as metrics) that potentially impact the fulfillment. Environmental factors (e.g., insufficient hardware, power shortage, or client demand) can thus be contrasted with the devices' capacity to fulfill SLOs. Based on that analysis, the leader reconfigures the cluster (e.g., by redistributing the load) so that QoS and QoE SLOs are optimized within the cluster.

3.3.1. Cluster-wide evaluation of SLOs

To analyze SLO fulfillment on a cluster level, the leader node does not reevaluate lower-level SLOs — this was already covered within the Edge. Instead, the leader node merely collects SLO compliance rates per device as a combined factor $f = pv \times ra$. These metrics are collected at the leader; depending on the desired amount of historical data, the high availability of the Fog would again be beneficial for collecting data. The question is now how to transfer metrics: Considering the potential size of a device cluster, we opt for a push-based approach, where devices periodically supply their data to the leader.

Apart from the SLO fulfillment, edge devices provide metrics that reflect their current environmental state. This includes any factors that the leader node should consider. If a battery-equipped device suffers occasional power shortages, it can report this conditional to the leader node, which adapts the network, e.g., by offloading computations to other devices to decrease its power drain. However, in the event of an entire network outage, devices can be incapable of reporting their state, and another node (e.g., leader) would have to detect this. Other frequent conditions can be general network congestion, including poor latency, jitter, or packet loss, but also devices' geographic location, user density, and peak usage times. Given their impact on the devices' capacity to fulfill SLOs, the leader node will rebalance the environment.

3.3.2. Analysis & optimization per device

Optimizing the devices' environments requires methods to draw conclusions between discrete environmental states and their consequential SLO fulfillment. To that extent, we aim – again – to identify causal relations between metrics; however, this time on a cluster level. Given a metric set (i.e., reflecting the environmental state) and the respective SLO rates per device, the leader node can construct a BN and infer how environmental changes impact the SLO fulfillment. To accelerate the construction of such a model, the leader node can combine metrics from devices of the same type, or even those that have comparable hardware characteristics (as done in Section 3.2.2). Although we ascended from an Edge to a cluster level, we still use the same tool for analyzing and adapting the environment — the EOSC model. However, to make a distinction, we call this new instance a EOSC-F (Fog) model.

Given a trained EOSC-F model (or rather, its BN), it is evident which environmental factors (σ_{env}) have a causal impact on SLO fulfillment. This can also help to improve the QoS in the long run, e.g., by pinpointing issues within the infrastructure. However, we aim to ensure SLO fulfillment the moment the QoS or QoE drops; the EOSC-F model can therefore consider the devices' environment and redistribute client load to ensure maximum SLO fulfillment within the cluster. To that extent, we present Algorithm 3, which distributes a number of streams (n_{client}) between the devices (Λ) in the cluster. `Inference` is again executed only on the variables that relate to SLO fulfillment, i.e., $\mathtt{MB}(model, f)$, by filtering the $model$ (Line 2 & 10). In Lines 6–18, the agent then iteratively assigns clients to the device, whose SLO fulfillment is the least impacted by receiving another stream ($ass[\lambda] + 1$). This assumes, that both ass and σ_{env} are part of ev, i.e., have an impact on SLO fulfillment. To that extent, $\sigma_{env}[\lambda]$ can contain factors like device characteristics. After assigning all streams within the cluster, the assignment can be orchestrated to the clients.

Algorithm 3 Client reassignment algorithm

```
Require: $model$, $n_{client}$, $\sigma_{env}$
Ensure: $ass$ // assignment according to env. state
 1: $i \leftarrow 0$
 2: $ev \leftarrow \mathtt{MB}(model, f)$
 3: for each $\lambda \in \Lambda$ do
 4:     $ass[\lambda] = 0$
 5: end for
 6: while $i < n_{clients}$ do
 7:     $\delta_{best} = -\infty$
 8:     for each $\lambda \in \Lambda$ do
 9:         $evidence \leftarrow ev \cap (\sigma_{env}[\lambda] \cup ass[\lambda] + 1)$
10:         $\delta \leftarrow \mathtt{INFERENCE}(ev, f, evidence)$
11:         if $\delta > \delta_{best}$ then
12:             $\delta_{best} = \lambda$
13:         end if
14:     end for
15:     $ass[\delta_{best}] \leftarrow ass[\delta_{best}] + 1$
16:     $i \leftarrow i + 1$
17: end while
18: return $ass$
```

3.3.3. Orchestration and redistribution

As a last step, the new cluster configuration must be enforced; in this case, by informing the pertinent devices of the new assignment. The leader node pushes this information to all edge devices that must alter their configuration. In accordance with Fig. 6, this includes all devices that offload or receive clients (red & blue); thus, the red device redirects clients to the blue device. To improve the SLO fulfillment rate within the cluster, the assignment considered each device's environment to provide an adequate configuration on a cluster level. Regardless of whether the QoS was impacted by poor network conditions or by poor hardware, if these conditions are packed as stateful information, the leader node can optimize the cluster accordingly. Thus, it covers heterogeneities between edge devices, which themselves might fail to scale their service given the stress introduced by the environment.

This concluded the client load redistribution, which optimized overall SLO fulfillment in the cluster according to the EOSC-F model. To transfer intelligence to the network edge, or even to the level of cluster or fog nodes, this section provided various concepts that all had the same goal: ensure SLOs in the respective system. It remains to provide a prototypical implementation of presented ideas, evaluate it according to key aspects, and argue to what extent it is ready for wider adoption.

4. Evaluation

In the following, we describe a CC scenario that requires edge devices to continuously transform video streams; this use case poses various requirements that must be ensured throughout processing. Afterward, we outline our prototype that ensures SLOs through collaborative edge intelligence. Essentially, this is the implementation of the presented framework. Lastly, we explain the methodology according to which the prototype will be evaluated. Section 5 will contain the respective results.

4.1. Use case description

The CC as a distributed system provides unprecedented opportunities for service providers and clients, e.g., in terms of processing or requirements assurance. As an example, consider a region with frequent natural disasters where the humanitarian situation should be documented. Therefore, reporters provide video streams in which vulnerable groups, e.g., minors of age, are detected. In the same step, individuals can be counted or visually highlighted; their identities, however, must be preserved. The region suffers from occasional network breakdowns (i.e., this affects access to global resources like the cloud but not internal connectivity); the reporting team thus provides ad hoc networking infrastructure in the form of edge devices, which are installed in close proximity to the operation area. Reporters equipped with IoT cameras are now capturing their surroundings; the video streams are transformed on edge devices, where they can be cached as long as global internet services are unavailable. Once resumed, videos are streamed to a cloud platform that provides the content to worldwide consumers.

Envisioned solution. Due to the nature of how disasters happen, it is impossible to fine-tune the complete streaming architecture beforehand. Therefore, the system is unaware of how to ensure its service (i.e., characterized by a set of SLOs) within this highly dynamic environment. To that extent, we advertise our framework for collaborative edge intelligence as the missing piece: Edge devices are supervised by AIF agents, which ensure QoS and QoE through their EOSC model. Whenever the computing architecture is extended with new devices (i.e., scaled horizontally), existing models can be transferred to this new device, regardless of whether its device type is known. Apart from that, the leader node continuously analyzes edge devices' capacity to comply with SLOs; in case some devices are excessively loaded or suffer from short-term network issues, the assignment between IoT cameras and edge devices is adapted to optimize the cluster-wide SLO fulfillment.

4.2. Implementation

While the last part of the use case outlined the envisioned solution, not all of these aspects are implemented and evaluated; in this regard, we focus on the ideas presented in this paper. This especially concerns the three contributions of the presented framework, i.e., the AIF-based model training, knowledge transfer between heterogeneous devices, and rebalancing of load according to environmental factors. Aspects such as bootstrapping of IoT and edge devices and leader node election (e.g., fog or edge) were already covered, e.g., by *Murturi et al.* [46,47]. The same applies to cloud-based distribution of video streams. An exception, however, are privacy-preserving stream transformations; for this, we make use of previously evaluated work [48]. To give our evaluation more rigor, we chose this over simulating a workload and its impact on SLOs.

4.2.1. Prototype

We provide the Python-based prototype of our framework in a GitHub repository[2]; it contains all source code we used to implement the three contributions, as well as the EOSC models for each device type. The core logic is separated into two classes: `Agent` and `FogNode`. These are the high-level loops executed in the main thread; all other processes (e.g., `AIF` or `VideoProcessor`) run in detached threads. The central library that is applied for training and updating BNs, as well as running inference queries, is *pgmpy* [49]. *pgmpy* offers ample support of BNL techniques; however our choice is also motivated by personal preference — the framework's performance must be analyzed under different libraries (e.g., as done by [50]). To improve the portability of our framework and simplify distribution, we provide a docker image[3] that can be executed platform independently.[4] The image exposes multiple env variables for configuring the solution, e.g., forcing the `Agent` to create a EOSC model from scratch or disabling `AIF` entirely.

The source code also contains the framework for privacy-preserving stream transformation and the ML models for face [51] and age detection [52]. To improve the reproducibility of results, we cancel out irregularities in the video streams by processing prerecorded videos; these are contained in the same repository. To simulate redirecting IoT devices within the cluster, it thus suffices to open/close processing threads on the edge devices; this simplifies networking. The `Agent` can thus reconfigure the stream assignment immediately, at the end of every `AIF` iteration. Because the use case is focused on video streaming and the number of frames per second (*fps*) that are transferred, each iteration lasts up to 1000 ms.

[2] https://github.com/borissedlak/workload/tree/main/FGCS

[3] https://hub.docker.com/repository/docker/basta55/workload/

[4] In fact the docker image is restricted to daemons with a *linux/arm64* architecture. This is the case for all device in Table 3, except for *Laptop* (*x86_64*).

Table 1
Device metrics captured, which are turned into model variables by AIF.

Name	Origin	Unit	Description	Param
pixel	IoT	num	Number of pixels contained in a frame	Edge
fps	IoT	num	Number of frames received per second	Edge
bitrate	IoT	num	Number of pixels transferred per second	No
cpu	Edge	%	Utilization of the device CPU	No
memory	Edge	%	Utilization of the system memory	No
streams	Edge	num	Number of IoT devices providing data	Fog
consumption	Edge	W	Energy pulled by the device	No
network	Edge	num	Network throughput per application	No
delay	App.	ms	Processing time per video frame	No
success	App.	T/F	If a pattern (i.e., face) was detected	No
distance	App.	num	Relative object movement between frames	No
slo_rate	Edge	%	Combined SLO Fulfillment rate ($pv \times ra$)	No
device_type	Edge	enum	Physical device type	No
congestion	Edge	num	Network congestion that increases latency	No

4.2.2. Practical limitations

Merging BN, as presented in Section 3.2.3, is only possible under the specified conditions, which are not always given during the AIF process. The number of states in a CPT, for example, is highly dynamic and extended as new batches of data are received. To merge the EOSC models under such circumstances, we provided a workaround: Instead of merging two BNs (m_a and m_b), we extend one of them (e.g., m_a). The device that trained m_b maintains a backup of the training data (d_b); this we use to update the CPTs of m_a through `PARL`,[5] i.e., $m_{ab} = \mathtt{PARL}(m_a, d_b)$. Notice, that this merges the conditional probabilities of the models, but not the structure; this remains an open question. While the resulting models are valid, we cannot assume that the original training data is always maintained.

Another limitation is that the DAG of the $model$ cannot be updated frivolously through `STRL`; this triggers numerous updates within the CPTs of the BN, which are not supported by default in *pgmpy*. Although *bnlearn* [53] promises these features, we require a package that can be embedded into our Python environment. Therefore, we make use of the following workaround: Instead of updating the DAG of model m_a according to new observations $batch$, we train a new BN with $data = batch \cup d_a$, where d_a reflects again the backup data. So internally, the AIF agent executes $\mathtt{STRL}(model, batch)$ as $\mathtt{PARL}(\mathtt{STRL}(data), data)$, which likewise updates the CPTs with every execution. Solving this limitation will be a far-reaching achievement that requires dedicated future work.

4.2.3. Variables and SLOs

For the given use case, the agents consider device and application (i.e., video processing) metrics to construct EOSC models. Internally, BNL transforms metrics into model variables, which are used to evaluate conditional probabilities. Table 1 contains an overview of all captured metrics; each row contains a description, measuring unit, and if it can be set as parameter. Notice, that only parameterizable variables can be adjusted by AIF agents to optimize SLO fulfillment. For example, *pixel* and *fps* are video stream properties of the IoT device, which are reconfigured by edge devices according to agents' behavior. The leader node, on the other hand, can adjust the number of *streams* per device, which is out of scope for individual devices.

The EOSC (or EOSC-F) models can be applied in different computational tiers to ensure each tier's unique requirements; thus, their model variables might not overlap. The edge-based EOSC model contains the upper part of the variables, i.e., from *pixel* to *success*, whereas the cluster-based EOSC-F model treats the lower part. Notice that the metric's origin, i.e., if it was measured from system stats or the application,

[5] This functionality is natively offered by *pgmpy*; by default, the models are merged proportionally to the number of samples that m_a and d_b contain. This can be fine-tuned by adjusting the *n_prev_samples* parameter; we use this to prioritize new observations $batch$ over existing conditional probabilities.

Table 2
Extracted SLOs and their classification.

SLO	Condition	Tier	Type
network	$throughput < 1.6$ MB/s	Edge	QoS
in_time	$delay < 1/\text{fps}$	Edge	QoS
success	$success = True$	Edge	QoE
distance	$distance < 50$	Edge	QoE
slo_rate	$\max(slo\_rate)$	Fog	Both

does not determine where it is used as a variable. From these variables, we construct SLOs that reflect the system state in terms of QoS and QoE. The AIF agent considers this classification when calculating pv and ra (recall). In Table 2, we present four SLOs that must be ensured during edge-based processing and one that is ensured by the cluster's leader node. To simplify the EOSC models, we include the SLO into BNL and remove the source variable, i.e., **distance** instead of *distance*.

We consider the presented SLOs relevant because (1) **network** ensures that the actual throughput does not exceed the bandwidth allocated to this application, (2) **in_time** makes sure that frames are computed within the available time frame, (3) **success** guarantees maximal privacy preservation, and (4) **distance** ascertains a smooth trajectory for tracked objects. The **slo_rate** reflects the cluster-wide SLO fulfillment. Notice that in the supplied video stream, there was always a face present, which means **success** can be compared against a ground truth.

4.2.4. Device classification

Video processing is very dependent on the availability of GPU acceleration [48]; therefore, we apply multiple edge devices — with and without GPUs. All devices applied for this work are listed in Table 3; in the following, we call them by their ID. The other columns contain hardware characteristics and – complementarily – the original price of the device. A special instance is $Xavier_{CPU}$: while its physical hardware is equal to $Xavier_{GPU}$, we disabled the GPU acceleration (i.e., NVIDIA CUDA) to create another device type. Overall, our devices differ greatly in terms of computing capabilities (e.g., missing GPU support or a highly superior CPU with 16 cores); nevertheless, as a whole, these devices compose the heterogeneous edge layer of the CC architecture.

As a prerequisite for transfer learning, we classify devices in a cluster according to their hardware characteristics. Although this process is dynamic, i.e., done repeatedly as devices join or leave the cluster, we focus our evaluation on a scenario where the cluster contains all devices from Table 3, excluding Xavier_{GPU}; the latter will be the device joining the cluster. As discussed in Section 3.2.2, we classify these devices relative to each other according to their CPU and GPU capabilities; the results are contained in Table 3. To achieve the desired distance between the scalars, the CPU is aligned between $[1 \leq p \leq 4]$ and the GPU between $[0 \leq g \leq 2]$.

4.3. Evaluation methodology

The implementation of the use case is thus set up for evaluation. To ensure a solid foundation for our framework, we will target each of the three pillars (i.e., the contributions) individually. The order in which they are evaluated resembles the one used throughout the paper; this makes sense also from a logical point of view because transfer learning and stream offloading rely on the underlying AIF mechanism. In the three paragraphs below, we outline the evaluated aspects and motivate each question. Combined, this represents our evaluation methodology.

Active inference. Our main interest includes the executability of the AIF agent on edge devices and the extent to which the EOSC model improves the SLO fulfillment within the Edge. Because structure and parameter learning are recurrent factors in the evaluation, we will put emphasis on when they happen. Namely, our questions include:

A-1: Do MBs reduce the complexity of inference?
Increasingly large BNs require mechanisms to limit the complexity of a system; otherwise, resource-restricted edge devices may fail to execute the AIF cycle within an induced time frame. The MB, as a potential remedy, could achieve this.
A-2: What is AIF's operational overhead?
Training and updating EOSC models directly on edge devices allows them to adapt quickly to system dynamics. However, any overhead introduced by AIF must not disrupt regular device operation, e.g., data processing.
A-3: How long require AIF agents to ensure SLOs?
To optimize SLO fulfillment, the agent must be able to infer adequate system configuration. However, there is no guarantee after how many AIF iterations the model will converge to the desired accuracy. Hence, we must provide an estimate for this.
A-4: Are the produced Bayesian networks interpretable?
Large-scale distributed systems, e.g., the CC, require trusted and reliable components as a solid foundation. Given that AIF can provide structures that are empirically verifiable, this promises to increase trust.
A-5: What is the operational impact of including BNL in the AIF cycle?
BNL was identified as the dominant factor for the complexity of the AIF cycle; therefore, we must ascertain whether edge devices can perform BNL without limitations. Depending on the results, the two processes could be broken up into a federated learning approach, e.g., to execute sub-steps in the Fog.
A-6: Can changes in variable distribution be handled?
Real-world generative processes are not guaranteed to stay stable, small environmental changes (e.g., a new client) might suffice to change the SLO result. Nevertheless, these changes should be detected and resolved through AIF-based model training.
A-7: Can SLOs be modified during runtime?
In the CC, devices can be administered by entities that stand hierarchically above them; these can change their role in the architecture, or more simply, their SLOs. If a device could not adapt its existing EOSC model, it would have to train from scratch.

Knowledge transfer. After focusing on the training of EOSC models, we are mainly interested in how well the created models can be exchanged with other edge devices, and if this promises to improve the training time. Ideally, we would thus reuse existing knowledge instead of "rediscovering" it.

K-1: What is the SLO fulfillment rate of transferred models?
Transfer learning can provide ML models (i.e., specific for one device) to other devices. However, it is not guaranteed that a transferred model performs equally to a model specifically trained for a device. For example, the transferred model might be more likely to violate SLOs.
K-2: Can knowledge transfer achieve any speedup?
Transferring a trained model removes computational overhead (*A-2*) from the recipient; thus, it could decrease the overall energy dedicated to model training, most beneficial for resource-restricted edge devices. Furthermore, this could decrease the time required to ensure SLOs (*A-3*).
K-3: Can merged models decrease the FE compared to choosing a single one?
Models with low FE can infer SLO-fulfilling system configurations with higher accuracy. Exchanging knowledge within the cluster can include the combination of multiple eligible models. However, can such combined models interpret observations with less surprise compared to a single transferred model?

Stream offloading. To optimize their SLO fulfillment, intelligent edge device continuously adapt their environment. However, for environmental factors that are out of their scope (e.g., network failures or hardware limitations), the device cluster can be the remedy to compensate for these issues. In this context, we want to determine whether the SLO fulfillment of individual devices can be recovered through collaboration.

Table 3
List of devices used for implementing and evaluating the presented methodology.

Full device name	ID	Price[a]	CPU	RAM	GPU	p [1,4]	g [0,2]	Σ
ThinkPad X1 Gen 10	*Laptop*	1800 €	Intel i7-1260P (16 core)	32 GB	Incompatible	Very high (4)	None (0)	4
Jetson Orin Nano	*Orin*	500 €	ARM Cortex A78 (6 core)	8 GB	Volta (383 core)	High (3)	High (2)	5
Nvidia Jetson Nano	*Nano*	150 €	ARM Cortex A57 (4 core)	4 GB	Incompatible	Low (1)	None (0)	1
Jetson Xavier NX	$Xavier_{CPU}$	300 €	ARM Carmel v8.2 (6 core)	8 GB	Disabled	Medium (2)	None (0)	2
Jetson Xavier NX	$Xavier_{GPU}$	300 €	ARM Carmel v8.2 (6 core)	8 GB	Amp (1024 core)	Medium (2)	Low (1)	3

[a] Price as of October 11th 2023 from https://sparkfun.com/.

S-1: How is load distributed among resource-constrained devices?
The Edge, as one CC tier, allows clients to request services from nearby edge devices; however, this fosters situations where load is highly unbalanced within the system. This might cause resource-restricted devices to fail their service; once this is detected, the load must be rebalanced within the system.
S-2: Can the CC hierarchy optimize local SLO fulfillment?
Depending on the scale of SLO failure, individual devices may be incapable of recovering their service through local reconfiguration. Nevertheless, higher entities in the CC (e.g., cluster) can evaluate and resolve this by employing their own SLOs.

5. Results and discussion

In the following, we evaluate the prototype according to the presented methodology. We structure our results according to the three contributions and the evaluation order in Section 4.3; based on the results, we pose derivative questions for future work. At the end of this section, we take a bird's-eye view to look at the results as one coherent framework and discuss the applicability of our approach.

5.1. Active inference

A-1: Do MBs reduce the complexity of inference? To show whether an MB can decrease the AIF cycle duration, we focus on one of its subparts — the inference. We modify the implementation of Algorithm 1 (Lines 2 & 8) to execute INFERENCE either (1) on the entire BN including all 4 SLOs, (2) the MB including 4 SLOs, (3) the MB with 2 SLOs, or (4) the MB with 1 SLO. Then, we execute the AIF cycle on *Laptop* and capture the running time of each configuration over a duration of 10 min; this produces 600 observations for each experiment. Fig. 7 visualizes the time that *Laptop* requires for performing INFERENCE, given the different MB sizes.

We observe: (1) applying an MB reduces the median execution type significantly, i.e., from 191 ms (gray) to 159 ms (blue) for 4 SLOs, and (2) decreasing the number of SLOs gradually reduces the execution time further. We thus conclude that MBs reduce the complexity of VE (*A-1*).

A-2: What is AIF's operational overhead? To evaluate AIF's overhead, we use pre-trained models for $Xavier_{CPU}$ and $Xavier_{GPU}$. Each device processes 6 video streams. We measure the CPU load (%) of the two devices with one of these two configurations: (1) AIF enabled, and (2) AIF disabled. We capture the load over 10 min; this produces 600 observations for each experiment. In Fig. 8, we show the CPU load of $Xavier_{CPU}$ and $Xavier_{GPU}$. The left bar of each device shows the load when operating with AIF and the right one without AIF.

We observe: (1) the CPU load is clearly decreased by videos processing on GPU, $Xavier_{GPU}$ with AIF enabled presented a 24% lower load than $Xavier_{CPU}$, and (2) the AIF background process introduced a computational overhead of 3% for both devices (left vs. right bar). Overall, this provides an estimate of the general overhead (*A-2*); however, whether this is acceptable depends on the use case.

A-3: How long require AIF agents to ensure SLOs? To evaluate the time to train a EOSC model, we count (1) the number of AIF iterations that the agent requires to arrive at a (nearly) optimal device configuration,

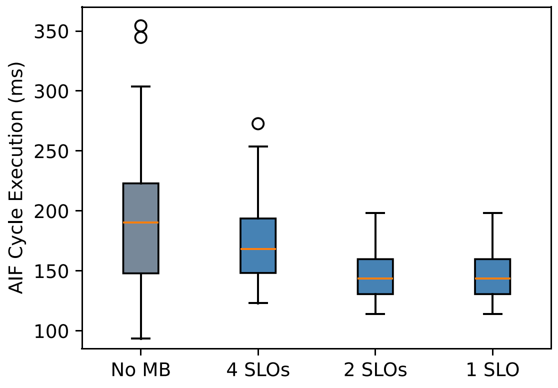


Fig. 7. Duration of AIF cycle depending on the application of an MB and the number of SLOs (*A-1*).

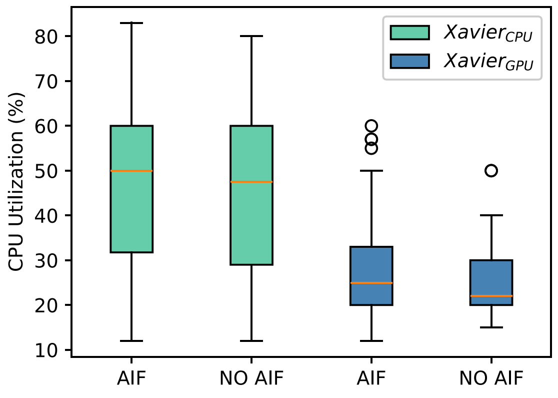


Fig. 8. Overhead introduced by AIF when operating on $Xavier_{CPU}$ or $Xavier_{GPU}$ (*A-2*).

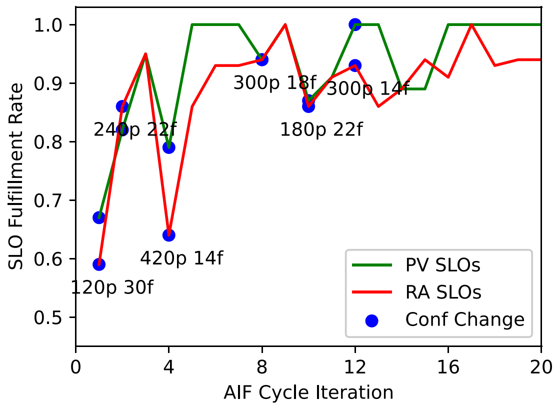


Fig. 9. SLO fulfillment rate (pv & ra) when operating on a blank *Laptop* client over 20 AIF cycles (*A-3*).

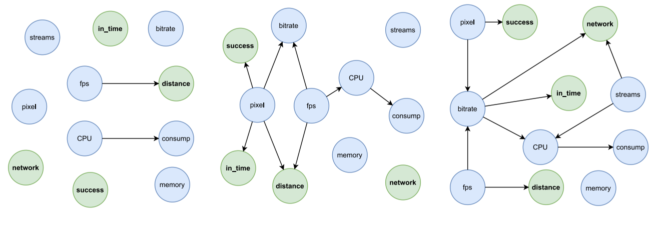


Fig. 10. Progress of the DAG after {1,3,10} rounds of parameter training when creating a model with AIF on *Laptop* (*A-4*).

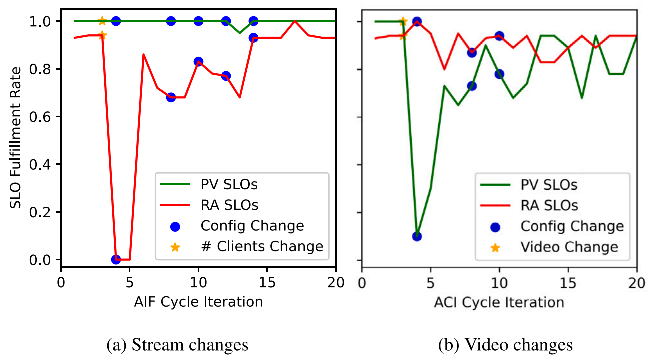


Fig. 11. Changes in the variable distribution caused (a) by higher number of video streams or (b) lower video quality (*A-6*).

and (2) how often the agent changes the configuration. The model is trained from scratch; therefore, the AIF agent (i.e., executed on *Laptop*) trains the model over 20 cycles and reports after each cycle (3) the SLO fulfillment according to the selected device configuration. We present the results in Fig. 9: The green and red lines represent the SLO fulfillment (pv & ra); whenever the agent reconfigures the edge device, we print a blue dot for both lines in the graph.

We observe: (1) the agent requires roughly 7 cycles to converge to a configuration that satisfied SLOs with more than 90%, which is maintained in later rounds; (2) this state is reached after 3 reconfigurations; and (3) pv and ra showed similar trends in this example. Thus, we answered how long an AIF agent requires to provide an acceptable configuration (*A-3*), both in terms of AIF cycles and the number of reconfigurations.

A-4: Are the produced causal graphs interpretable? To discuss the interpretability of created causal structures, we compare the DAGs produced by STRL and highlight at which stage the graph can be empirically explained. We will not consider specific metrics here but interpret the DAGs according to our expert knowledge. On *Laptop*, we train a EOSC model from scratch and extract the DAGs after {1,3,5,10} rounds of BNL. Thus, we want to show how the AIF agent discovers (ideally) causal relations between model variables. The results are visible in Fig. 10: SLO variables (see Table 2) are colored in green; regular variables in blue.

We observe: (1) all SLO variables are influenced by variables that the AIF agent can control, and (2) memory was the only variable that could not be related to others. After studying the graphs carefully, we could not detect any edge that appears counterintuitive to us; however, this does not prove that they are indeed causal. In total, we claim that the created graph is coherent and the links are understandable (*A-4*), but it requires sophisticated experiments to prove causality for each edge.

A-5: What is the operational impact of including BNL in the AIF cycle? To answer whether BNL can be applied on regular edge devices, we train a EOSC model on $Xavier_{GPU}$ and measure the execution time of STRL and PARL, i.e., the BNL sub-steps from Algorithm 2. In Fig. 12 we visualize the execution time of STRL and PARL over 100 AIF iterations, respectively 1.5 min of operation. We observe: (1) PARL requires a stable runtime of around 250 ms, (2) the runtime of STRL increases as more training data becomes available, and (3) running STRL after 100 AIF iterations took more than 20 s. We conclude that PARL might be run on the employed edge device because it can be completed within less than 1000 ms (i.e., the time frame for concluding the AIF cycle from Section 4.2.1). However, the runtime of STRL presents an obstacle because the AIF agent might thus have to skip iterations until the ongoing execution of STRL finishes. Hence, it would be advisable to perform STRL on another device (*A-5*) or find a way to decrease the runtime, e.g., by updating the DAG regardless of existing CPTs.

A-6: Can changes in variable distribution be handled? Variable distributions can change due to various external factors; to evaluate how well the system can handle this, we either (1) simulate a peek usage time by increasing the number of processed video streams from 1 to 6, or (2) distort the video content with a Gaussian blur of 5px, which could resemble a foggy video setting. We measure the impact on the SLO fulfillment (pv & ra) over 20 AIF cycles and visualize to what extent the EOSC model is capable of restoring satisfactory (i.e., close to original) SLO rates. Fig. 11 shows in both subfigures the SLO fulfillment rate of *Laptop*, when the disruptive factor was introduced (i.e., after 3 iterations), and at which points the AIF agent reconfigured the system (blue dots).

We observe: (1) after the stream change, *Laptop* took 11 AIF cycles (incl. 4 reconfigurations) to recover the SLO fulfillment, and (2) the information loss introduced by the video manipulation could not be recovered, although SLO fulfillment was improved as far as possible. Hence, we conclude that the system was able to adapt to changes in the variable distribution (*A-6*); however, only as long as the device can compensate for this factor. In fact, the **success** SLO could not be fulfilled after the video change took place because the agent could not increase the resolution sufficiently to recognize the faces.

A-7: Can SLOs be modified during runtime? To simulate changing requirements, we modify the **distance** SLO from 50 to 20 (i.e., clearly stricter) and measure the SLO fulfillment rate before and after the modification. Additionally, we capture the surprise (Algorithm 1) to show if SLO outcomes reflected the expectations of the agent. Fig. 13 shows in the upper part the SLO fulfillment rate over 40 AIF cycles; the SLO changes after 3 iterations. The lower part shows the agent's surprise at each round and when STRL or PARL happen.

We observe: (1) after the SLO change, the agent experienced 9 rounds of high surprise, i.e., $\gg$ 35, (2) after 2 reconfigurations, the state prior to the SLO change was recovered, although final SLO rates (mean 0.91) are slightly below previous (mean 0.94), (3) to satisfy lower **distance**, the answer was to increase *fps*, and (4) the magnitude of the surprise was decisive for the decision between STRL and PARL (as envisioned in Algorithm 2). However, as known from Fig. 12, STRL can exceed the AIF time frame multiple times; hence, the AIF agent is forced to wait for this process to finish. This could be solved, e.g., by offloading STRL. Hence, we conclude that the system was able to handle SLO changes during runtime (*A-7*).

5.2. Knowledge transfer

K-1: What is the SLO fulfillment rate of transferred models? Transfer learning promises to accelerate model training, but we must ascertain that transferred models perform equally to trained ones. For this, we assume that $Xavier_{GPU}$ wants to join the device cluster. According to Table 3, *Laptop* and $Xavier_{CPU}$ are eligible for providing their model, i.e., their dc (2 & 4) are the closest to $Xavier_{GPU}$ (3). Hence, we merge their EOSC models and transfer the result to $Xavier_{GPU}$. Next,

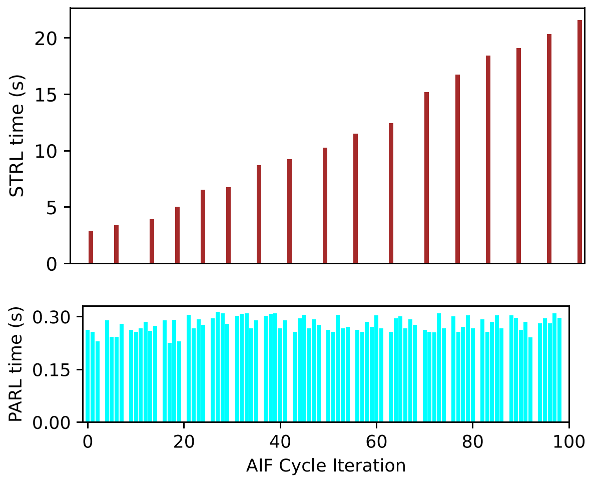


Fig. 12. Duration of structure and parameter learning on $Xavier_{GPU}$ when training a BN from scratch (*A-5*)

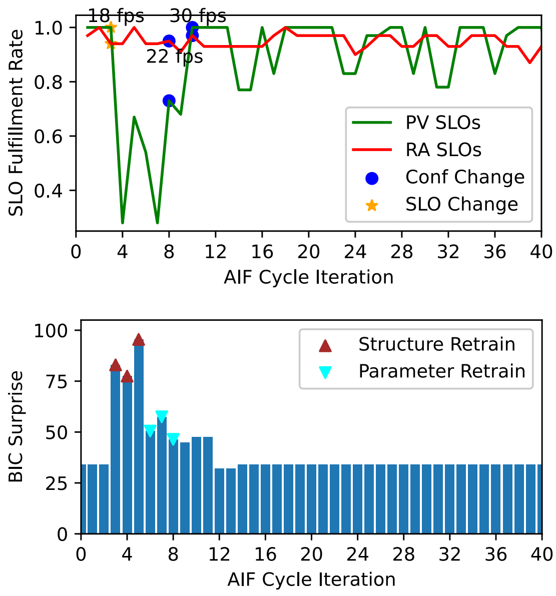


Fig. 13. Impact of changing the **distance** SLO during runtime, combined with the surprise measured (*A-7*).

we compare the SLO fulfillment of the merged model with a separate run, where a model is trained from scratch. We place both runs into Fig. 14; the blue line represents the combined model, and the gray one was trained from scratch. Additionally, we indicate each time the agents changed the configuration.

We observe: (1) the merged model does not face any substantial improvements of its initially high SLO fulfillment; (2) the agent required 14 rounds to arrive at a comparable SLO rate — this also matches our experience from Fig. 9, where *Laptop* required 7 to 16 AIF rounds for training; and (3) the final rates are within the range [0.85,0.95]. From that, we conclude that the results produced by the trained model were comparable to the merged model (*K-1*), and that KT could achieve a speedup of 14 rounds (*K-2*), assuming that the transferred model was ready for usage. Nevertheless, this is only valid for the given setup (i.e., these two devices); it is not possible yet to derive general implications of our approach.

K-3: Can merged models decrease the FE compared to choosing a single one? As discussed in Section 2.1.2, it is hard to estimate the FE of a model, but we consider the fact that surprise is bounded by FE. Although low

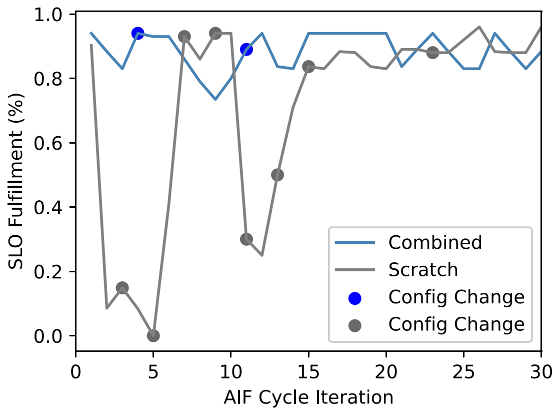


Fig. 14. Difference in SLO fulfillment between an agent using a transferred model or training from scratch (*K-1* & *K-2*).

surprise does not imply low FE, we use it as an indicator: We transfer a model to $Xavier_{GPU}$ (merged from *Laptop* and $Xavier_{CPU}$ as above) and calculate the surprise throughout multiple AIF cycles. This we compare against alternative runs, in which $Xavier_{GPU}$ uses one of the EOSC models of the other devices (from Table Table 3). Furthermore, we count the usage of PARL. The results are presented in Fig. 15; each of the colored lines represents one of the respective models, which were copied to $Xavier_{GPU}$. The blue line, however, describes the combined model. The lower figure shows for each run when PARL was executed.

We observe: (1) the models trained on *Orin* and *Nano* produced initially very high surprise ($\gg$50), indicating that these models fit $Xavier_{GPU}$ the least; (2) nevertheless, the agent was able to improve these models and converge to an area where all 5 models provide similar surprise after 25 iterations; (3) the combined model provided initially the best values and only performed PARL twice; and (4) interestingly, although close to each other, the combined model produces after 25 rounds the highest surprise (33), while $Xavier_{CPU}$ reached 17. This shows, that the frequent retraining performed by the other devices (colored triangles in the lower graph) allowed the other models to surpass $Xavier_{GPU}$. This raises the question if it would be advisable to always run PARL, regardless of the surprise magnitude Combined, we can answer that the merged model had initially less surprising values (*K-3*); however, frequent retraining may achieve even better results.

5.3. Stream offloading

S-1: How is the load distributed among resource-constrained devices? To offload computations within the cluster, we aim to show how low-resource devices are relieved from excessive load. For this, we assume 25 IoT devices that are either assigned *Equal* to the edge devices or *Random*. As an indicator for maximum SLO fulfillment, we added *Single*, where each device processes one stream; Table 4 shows an overview of each scenario's assignment. After operating with *Equal* or *Random*, the leader node starts to optimize the environment, i.e., using the EOSC-F model to distribute the 25 streams depending on the device capabilities (*Infer*). This new assignment is then provided to the edge devices. We thus simulated an offloading or load rebalancing, e.g., *Nano* dropped from 5 (or 3) to 1 stream. In Fig. 17, we show each device's SLO fulfillment rate per scenario. The left bars of Fig. 17(b) show the cluster-wide average of the SLO fulfillment and the right bar the weighted average according to the number of streams ($slo\_rate \times stream$). To get a feeling of the heterogeneous device capabilities, Fig. 16 provides a regression function that shows how the SLO fulfillment per device is impacted by the number of *streams*. We observe: (1) the average SLO fulfillment clearly improved by using *Infer* (0.81) instead of *Random* (0.64) or *Equal* (0.60); (2) this is also reflected by the weighted average

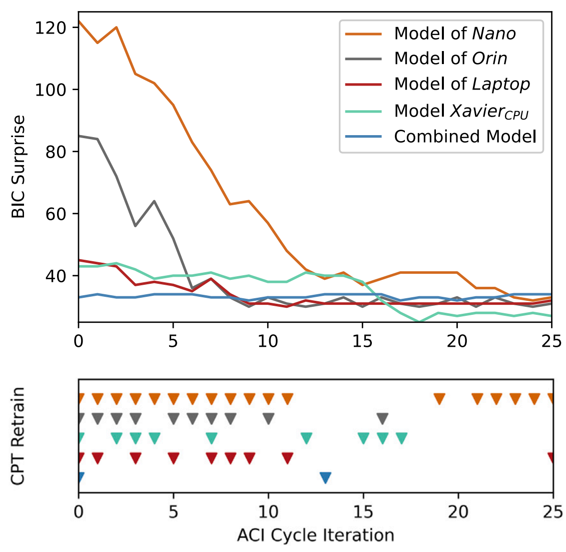


Fig. 15. Surprise per batch when operating on $Xavier_{GPU}$ with a combined or existing model. Paired with the frequency of PARL (*K-3*).

Table 4
Streams for scenarios.

Device ID	*Single*	*Equal*	*Rand*	*Infer*
Laptop	1	5	4	9
$Xavier_{GPU}$	1	5	8	5
$Xavier_{CPU}$	1	5	5	1
Orin	1	5	4	9
Nano	1	5	3	1
Sum Σ	5	25	25	25

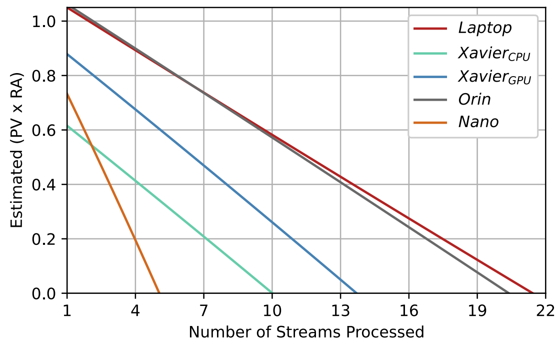
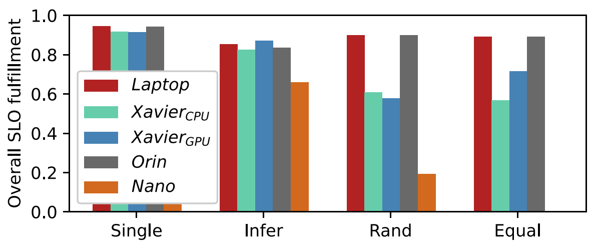


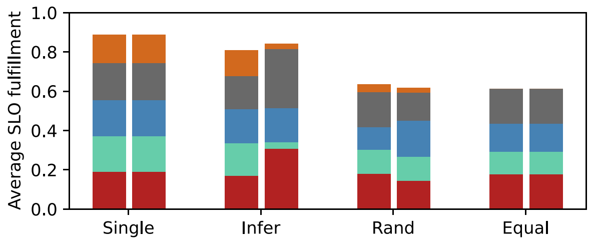
Fig. 16. Regression between *streams* assigned to edge devices and respective SLO fulfillment rate ($pv \times ra$).

(right bars of Fig. 17(b)), which puts *Laptop* and *Orin* in focus that processed 9 streams each; (3) the weighted average of *Infer* comes close to *Single* (0.89), even though the cluster processed 25 instead of only 5 streams. From that, we conclude that the intelligent cluster was able to incorporate restricted edge devices (e.g., *Nano*) into the architecture (*S-1*), and that the overall SLO compliance improved by following our approach.

S-2: Can the CC hierarchy optimize local SLO fulfillment? To improve the SLO fulfillment whenever individual devices lack the required scope, we will resolve such SLO failures within the cluster. Therefore, we consider a condensed device cluster consisting of *Laptop* and *Orin*. *S-1* showed that they have comparable processing capabilities; therefore, it is fair to split 10 streams equally between them. Fig. 18(b) provides the DAG internal to the EOSC-F model: Blue nodes are environmental factors, from which only *stream* can be configured (recall Section 4.2.3);



(a) SLO fulfillment per device



(b) Average and weighted average per batch

Fig. 17. SLO fulfillment within the edge–fog cluster when distributing load according to *Infer*, *Random*, or *Equal*. *Single* is an upper bar for this device constellation (*S-1*)

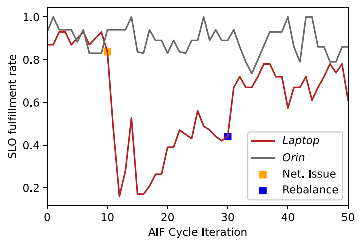
slo_rate represents the common factor $f = pv \times ra$. We simulate network congestion[6] for *Orin* – which the leader node can evaluate through *congestion* – and redistribute the load according to the EOSC-F model, i.e., $Orin = 8$, $Laptop = 2$. Then, we compare the overall SLO fulfillment before and after offloading; the results are shown in Fig. 18(a). The two lines show the SLO fulfillment (f) of *Laptop* (red) and *Orin* (blue) over 50 AIF iterations; after 10 rounds, the network gets congested. In round 30, the cluster leader rebalanced the load according to its EOSC-F model; although it is possible to rebalance earlier, we decided to observe the system behavior until manually rebalancing in round 30.

We observe: (1) the network issue crushed the SLO fulfillment of *Laptop* from around 0.9 to a minimum of 0.2 at round 15; (2) the edge device was able to improve the rate in the following 20 iterations by reconfiguration, until reaching a local optimum at 0.43. Further, (3) the cluster-wide SLO compliance was clearly improved through rebalancing, i.e., at round 15 the sum of $f_{Laptop} + f_{Orin}$ was 1.03, at round 30 it was 1.33, while at round 45 it rose to 1.54. We conclude that the intelligent cluster was able to resolve the introduced network issue (*S-2*) by redistributing the load according to the EOSC-F model. However, to draw general conclusions, we aim to consider a larger range of potential issues.

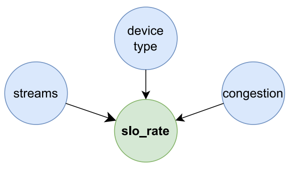
5.4. Summary and implications

As a summary, we can report that (1) edge devices were gradually able to ensure local SLO compliance without prior knowledge; it took them 16 rounds to identify factors that impact SLO fulfillment and adapt the environment accordingly; the resulting SLO fulfillment aligns close to existing work [30], (2) the underlying causal structures and the transitions between device configuration were empirically explainable; this increases traceability and trust of ML models, and (3) shifted variable distributions were canceled out through continuous model retraining; edge devices took 9 rounds to interpret an unprecedented increase in demand, while SLO failures introduced by poor video quality could not be fully recovered. Further, (4) the causality filter

[6] Internally, we increase the processing delay according to *congestion*; this increases the overall latency and causes **in_time** to fail more likely. The EOSC-F model considers *congestion* as environmental factor for Algorithm 3.



(a) Impact of rebalancing



(b) Internal variable relations

Fig. 18. Recovering network congestion by rebalancing the load within the device cluster according to the EOSC-F model; both devices initially processed 5 streams, 3 are offloaded to *Orin* (*S-2*)

based on MBs decreased the complexity of inference and sped up SLO evaluation by 17%, and (5) our framework introduced a negligible CPU overhead of 3%, which makes it a suitable choice for resource-restricted devices.

It turned out that (6) BNL, or in particular structure learning, surpassed the given time frame for continuous model adaptation; nevertheless, parameter learning took only less than 250 ms and the overall training time appears promising compared to [54]. Thus, (7) models transferred between nearby devices could be continuously improved, even in cases where they fit poorly; this improves the reusability of models in the heterogeneous Edge, (8) the SLO fulfillment of devices with transferred models equaled the one of self-trained models; this accelerated the distribution of SLO-compliance models within one computational tier by up to 16 rounds, (9) rebalancing the load after a network error increased the overall SLO fulfillment from 1.03 to 1.54; this showed that collaboration within this tier increased the scope of SLO failures that could be covered. A closing observation is that (10) variable shifts showed the same effects on SLO fulfillment as low accuracy after transferring a model to an unknown device type. To our framework, they did not provide any fundamental differences, which is why they could both be resolved through continuous model training.

6. Conclusion and future work

This paper presented a novel framework for collaborative and distributed edge intelligence that ensures decentralized SLO fulfillment. It allows CC systems to disaggregate high-level requirements and enforce them at the component they concern; thereby, we create self-adaptive devices that themselves ensure dynamic requirements. For each component, the framework is able to develop causal reasoning between environmental factors and SLO fulfillment. Resource-restricted devices that cannot create this knowledge were able to exchange and combine causal models according to their hardware characteristics. This accelerates the onboarding of unknown device types and simplifies horizontal scaling within the Edge. Contrarily, any attempt to achieve this centrally would struggle with heterogeneous device characteristics, the induced network latency, and the communication overhead. To increase SLO coverage and the action scope, devices collaborated as clusters under the supervision of a Fog node; this forms higher-level components that can again supervise their own set of SLOs. Consequentially, the cluster was able to use its extended environment to resolve SLO violations, e.g., by offloading computation among pertinent devices. Erecting these hierarchical structures provides an accurate representation of observable processes and infers how to fulfill the intricate requirements of multiple computational tiers.

We provided a prototype of the framework for a distributed video transformation use case and evaluated it according to 12 aspects; the results showed the potential of our approach for ensuring SLOs throughout CC tiers. For future work, we aim to dynamically update the structure of presented models and evaluate limitations regarding the number of SLOs and devices. Further, this work builds heavily on (causal) relations between SLO fulfillment and environmental factors; however, to prove causality, dedicated experiments must be integrated into the framework. Once this is established, the framework will provide necessary causal links to tame requirements in the CC.

CRediT authorship contribution statement

Boris Sedlak: Writing – original draft, Visualization, Software, Methodology, Data curation, Conceptualization. **Victor Casamayor Pujol:** Writing – review & editing, Validation, Investigation. **Praveen Kumar Donta:** Formal analysis. **Schahram Dustdar:** Supervision.

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.

Data availability

We provide all the data used for this paper, as cited within the article.

Acknowledgments

Funded by *European Union* (*TEADAL, 101070186*). Views and opinions expressed are those of the authors and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible.

References

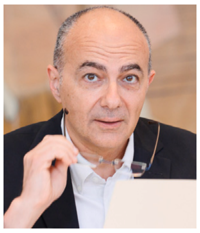
[1] P. Beckman, et al., Harnessing the computing continuum for programming our world, in: Fog Computing, John Wiley & Sons, Ltd, 2020.
[2] S. Dustdar, V.C. Pujol, P.K. Donta, On distributed computing continuum systems, IEEE Trans. Knowl. Data Eng. 35 (4) (2023) 4092–4105, http://dx.doi.org/10.1109/TKDE.2022.3142856.
[3] W. Tärneberg, et al., The 6G Computing Continuum (6GCC): Meeting the 6G computing challenges, in: International Conference on 6G Networking, 2022.
[4] V. Casamayor-Pujol, A. Morichetta, I. Murturi, P.K. Donta, S. Dustdar, Fundamental research challenges for distributed computing continuum systems, Information 14 (2023) 198, http://dx.doi.org/10.3390/info14030198.
[5] B. Sedlak, V.C. Pujol, P.K. Donta, S. Dustdar, Designing reconfigurable intelligent systems with Markov blankets, in: Service-Oriented Computing, 2023, pp. 42–50, http://dx.doi.org/10.1007/978-3-031-48421-6_4.
[6] K. Friston, L. Da Costa, N. Sajid, C. Heins, K. Ueltzhöffer, G.A. Pavliotis, T. Parr, The free energy principle made simpler but not too simple, 2023, http://dx.doi.org/10.48550/arXiv.2201.06387.
[7] H. Kokkonen, L. Lovén, N.H. Motlagh, A. Kumar, J. Partala, T. Nguyen, V.C. Pujol, P. Kostakos, T. Leppänen, A. González-Gil, E. Sola, I. Angulo, M. Liyanage, M. Bennis, S. Tarkoma, S. Dustdar, S. Pirttikangas, J. Riekki, Autonomy and intelligence in the computing continuum: Challenges, enablers, and future directions for orchestration, 2023, http://dx.doi.org/10.48550/arXiv.2205.01423.
[8] J. Pearl, Causal inference in statistics: An overview, Stat. Surv. 3 (none) (2009) 96–146, http://dx.doi.org/10.1214/09-SS057.
[9] N. Ganguly, D. Fazlija, M. Badar, M. Fisichella, S. Sikdar, J. Schrader, J. Wallat, K. Rudra, M. Koubarakis, G.K. Patro, W.Z.E. Amri, W. Nejdl, A review of the role of causality in developing trustworthy AI systems, 2023, http://dx.doi.org/10.48550/arXiv.2302.06975.
[10] P. Chen, Y. Qi, D. Hou, CauseInfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment, IEEE Trans. Serv. Comput. (2019).
[11] J. Lin, P. Chen, Z. Zheng, Microscope: Pinpoint performance issues with causal graphs in micro-service environments, in: C. Pahl, M. Vukovic, J. Yin, Q. Yu (Eds.), Service-Oriented Computing, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 3–20.
[12] I. Tsamardinos, C.F. Aliferis, A. Statnikov, Time and Sample Efficient Discovery of Markov Blankets and Direct Causal Relations, Association for Computing Machinery, New York, USA, 2003.
[13] A. Niculescu-Mizil, R. Caruana, Inductive transfer for Bayesian network structure learning, in: Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, PMLR, 2007, pp. 339–346.

[14] M.J. Vowels, N.C. Camgoz, R. Bowden, D'ya like DAGs? A survey on structure learning and causal discovery, 2021.
[15] V.C. Pujol, P. Raith, S. Dustdar, Towards a new paradigm for managing computing continuum applications, in: 2021 IEEE Third International Conference on Cognitive Machine Intelligence, CogMI, 2021.
[16] K. Friston, Life as we know it, J. R. Soc. Interface 10 (86) (2013) 20130475, http://dx.doi.org/10.1098/rsif.2013.0475.
[17] M. Kirchhoff, T. Parr, E. Palacios, K. Friston, J. Kiverstein, The Markov blankets of life: autonomy, active inference and the free energy principle, J. R. Soc. Interface (2018).
[18] K.J. Friston, J. Daunizeau, S.J. Kiebel, Reinforcement learning or active inference? PLOS ONE 4 (7) (2009) e6421.
[19] R. Smith, K.J. Friston, C.J. Whyte, A step-by-step tutorial on active inference and its application to empirical data, J. Math. Psych. 107 (2022) 102632, http://dx.doi.org/10.1016/j.jmp.2021.102632.
[20] N. Sajid, P.J. Ball, T. Parr, K.J. Friston, Active inference: demystified and compared, Neural Comput. 33 (3) (2021) 674–712.
[21] T. Parr, G. Pezzulo, K.J. Friston, Active Inference: The Free Energy Principle in Mind, Brain, and Behavior, The MIT Press, 2022.
[22] G. Camps-Valls, A. Gerhardus, U. Ninad, G. Varando, G. Martius, E. Balaguer-Ballester, R. Vinuesa, E. Diaz, L. Zanna, J. Runge, Discovering causal relations and equations from data, Phys. Rep. 1044 (2023) 1–68, http://dx.doi.org/10.1016/j.physrep.2023.10.005.
[23] E.C. Martínez, J.W. Kim, T. Barz, M. Cruz, Probabilistic modeling for optimization of bioreactors using reinforcement learning with active inference, Comput. Aided Chem. Eng. (2021).
[24] A. Tschantz, B. Millidge, A.K. Seth, C.L. Buckley, Reinforcement learning through active inference, 2020.
[25] C. Heins, B. Millidge, D. Demekas, B. Klein, K. Friston, I. Couzin, A. Tschantz, pymdp: A Python library for active inference in discrete state spaces, J. Open Source Softw. (2022).
[26] B. Sedlak, V.C. Pujol, P.K. Donta, S. Dustdar, Active inference on the edge: A design study, in: 2024 IEEE PerCom Workshops, 2024, pp. 550–555, http://dx.doi.org/10.1109/PerComWorkshops59983.2024.10502828.
[27] G. Levchuk, K. Pattipati, D. Serfaty, A. Fouse, R. McCormack, Active inference in multiagent systems: Context-driven collaboration and decentralized purpose-driven team adaptation, in: Artificial Intelligence for the Internet of Everything, Academic Press, 2019.
[28] B. Sudharsan, J.G. Breslin, M.I. Ali, Edge2Train: a framework to train machine learning models (SVMs) on resource-constrained IoT edge devices, in: Proceedings of the 10th International Conference on the Internet of Things, IoT '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–8, http://dx.doi.org/10.1145/3410992.3411014.
[29] W. Hao, Z. Wang, L. Hong, L. Li, N. Karayanni, C. Mao, J. Yang, A. Cidon, Monitoring and adapting ML models on mobile devices, 2023, http://dx.doi.org/10.48550/arXiv.2305.07772.
[30] Z. Zhang, Y. Zhao, J. Liu, Octopus: SLO-aware progressive inference serving via deep reinforcement learning in multi-tenant edge cluster, in: Service-Oriented Computing, Cham, 2023, http://dx.doi.org/10.1007/978-3-031-48424-7_18.
[31] V. Goyal, R. Das, V. Bertacco, Hardware-friendly user-specific machine learning for edge devices, ACM Trans. Embed. Comput. Syst. 21 (5) (2022) 62:1–62:29, http://dx.doi.org/10.1145/3524125.
[32] Q. Wu, H. Wu, X. Zhou, M. Tan, Y. Xu, Y. Yan, T. Hao, Online transfer learning with multiple homogeneous or heterogeneous sources, IEEE Trans. Knowl. Data Eng. 29 (7) (2017) 1494–1507, http://dx.doi.org/10.1109/TKDE.2017.2685597.
[33] Y.-C. Hsu, Z. Lv, Z. Kira, Learning to cluster in order to transfer across domains and tasks, in: Sixth International Conference on Learning Representations, ICLR 2018, 2018, http://dx.doi.org/10.48550/arXiv.1711.10125.
[34] S. Dustdar, Y. Guo, B. Satzger, H.-L. Truong, Principles of elastic processes, IEEE Internet Comput. 15 (2011) 66–71.
[35] S. Nastic, A. Morichetta, T. Pusztai, S. Dustdar, X. Ding, D. Vij, Y. Xiong, SLOC: Service level objectives for next generation cloud computing, IEEE Internet Comput. 24 (3) (2020).
[36] J. Fürst, M. Fadel Argerich, B. Cheng, A. Papageorgiou, Elastic services for edge computing, in: 2018 14th International Conference on Network and Service Management, CNSM, 2018, pp. 358–362.
[37] V.H. Menino, A Novel Approach to Load Balancing in P2P Overlay Networks for Edge Systems, 2021.
[38] M. Scanagatta, A. Salmerón, F. Stella, A survey on Bayesian network structure learning from data, Prog. Artif. Intell. 8 (2019).
[39] C. Aliferis, et al., Local causal and Markov blanket induction for causal discovery and feature selection part I: Algorithms and empirical evaluation, J. Mach. Learn. Res. 11 (2010).
[40] V. Casamayor Pujol, P. Raith, S. Dustdar, Towards a new paradigm for managing computing continuum applications, in: IEEE 3rd International Conference on Cognitive Machine Intelligence, CogMI 2021, 2021.
[41] N. Zhang, D. Poole, A simple approach to Bayesian network computations, in: Engineering-Economic Systems, Stanford, 1994.
[42] D. Ghio, A.L.M. Aragon, I. Biazzo, L. Zdeborová, Bayes-optimal inference for spreading processes on random networks, Phys. Rev. E 108 (4) (2023) 044308, http://dx.doi.org/10.1103/PhysRevE.108.044308.
[43] V.C. Pujol, A. Morichetta, S. Nastic, Intelligent sampling: A novel approach to optimize workload scheduling in large-scale heterogeneous computing continuum, in: 2023 IEEE International Conference on Service-Oriented System Engineering, SOSE, 2023.
[44] M. Vagnoli, R. Remenyte-Prescott, Updating conditional probabilities of Bayesian belief networks by merging expert knowledge and system monitoring data, Autom. Constr. 140 (2022) 104366.
[45] M. Vaniš, Z. Lokaj, M. Šrotýř, A novel algorithm for merging Bayesian networks, Symmetry 15 (7) (2023) 1461, http://dx.doi.org/10.3390/sym15071461.
[46] I. Murturi, S. Dustdar, A decentralized approach for resource discovery using metadata replication in edge networks, IEEE Trans. Serv. Comput. 15 (5) (2022) 2526–2537.
[47] S. Dustdar, I. Murturi, Towards distributed edge-based systems, in: 2020 IEEE Second International Conference on Cognitive Machine Intelligence, CogMI, IEEE, Atlanta, GA, USA, 2020, pp. 1–9.
[48] B. Sedlak, I. Murturi, P.K. Donta, S. Dustdar, A privacy enforcing framework for transforming data streams on the edge, IEEE Trans. Emerg. Top. Comput. (2023) http://dx.doi.org/10.1109/TETC.2023.3315131.
[49] A. Ankan, J. Textor, pgmpy: A Python toolkit for Bayesian networks, 2023.
[50] Q. Zhang, X. Che, Y. Chen, X. Ma, M. Xu, S. Dustdar, X. Liu, S. Wang, A comprehensive deep learning library benchmark and optimal library selection, IEEE Trans. Mob. Comput. (2023) 1–14.
[51] Linzaer, Ultra fast face-detector, 2022, https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB.
[52] R. Rothe, R. Timofte, L.V. Gool, DEX: Deep expectation of apparent age from a single image, in: 2015 IEEE International Conference on Computer Vision Workshop, ICCVW, IEEE, Santiago, Chile, 2015, pp. 252–257, http://dx.doi.org/10.1109/ICCVW.2015.41.
[53] M. Scutari, Learning Bayesian networks with the bnlearn R package, J. Stat. Softw. 35 (2010) 1–22.
[54] R. Kolcun, D.A. Popescu, V. Safronov, P. Yadav, A.M. Mandalari, Y. Xie, R. Mortier, H. Haddadi, The case for retraining of ML models for IoT device identification at the edge, 2020, http://dx.doi.org/10.48550/arXiv.2011.08605.

Boris Sedlak is currently working as a Ph.D. student at the Distributed Systems Group at TU Wien, Austria. He received his B.Sc. in Media Informatics at the University of Applied Sciences in St. Pölten, and his M.Sc. in Software Engineering & Internet Computing at the TU Wien. He worked for four years in the field of software engineering before focusing on his doctoral studies; his research interests include edge intelligence, causal methods for the computing continuum, and service oriented computing.

Victor Casamayor Pujol is a project assistant in the Distributed Systems Group at Technische Universität (TU) Wien, 1040, Vienna, Austria. His research interests revolve around self-adaptive methodologies for computing continuum systems, including SLO-based definitions, causal and machine learning inference, and robotics. Casamayor Pujol received his Ph.D. degree in information and communication technologies from Universitat Pompeu Fabra, Barcelona, Spain in 2020. He is a Member of IEEE.

Praveen Kumar Donta currently working as a Postdoctoral researcher at Distributed Systems Group, TU Wien, Austria. He received his Ph.D. at the Indian Institute of Technology (Indian School of Mines), Dhanbad from the Department of Computer Science and Engineering in May 2021. From July 2019 to Jan 2020, he is a visiting Ph.D. fellow at Mobile & Cloud Lab, Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia, under the Dora plus grant provided by the Archimedes Foundation, Estonia. He received his Master's in Technology and Bachelor's in Technology from the Department of Computer Science and Engineering at JNTUA, Ananthapur, with Distinctions in 2014 and 2012. He is serving as an Editorial Board member for Computer Communications, Measurement, Measurement: Sensors from Elsevier, PLOS ONE, Wireless Communications and Mobile Computing, and Hindawi, and ETT Wiley journals. His current research includes the Learning algorithms for Sensor Networks, Internet of Things, Computing Continuum.



Schahram Dustdar is Full Professor of Computer science heading the Research Division of Distributed Systems at the TU Wien, Austria. He was founding Co-Editor-in-Chief of ACM Transactions on Internet of Things (ACM TIoT) and is Editor-in-Chief of Computing (Springer). He is an Associate Editor of IEEE Transactions on Services Computing, IEEE Transactions on Cloud Computing, ACM Transactions on the Web, and ACM Transactions on Internet Technology, as well as on the editorial board of IEEE Internet Computing and IEEE Computer. Dustdar is Recipient of the ACM Distinguished Scientist Award (2009), the ACM Distinguished Speaker award (2021), the IBM Faculty Award (2012), an Elected Member of the Academia Europaea: The Academy of Europe, where he served as Chairman of the Informatics Section, as well as an IEEE Fellow. In 2021 Dustdar was elected President for Asia-Pacific Artificial Intelligence Association (AAIA).