DISTRIBUTED INTELLIGENCE IN THE COMPUTING CONTINUUM WITH ACTIVE INFERENCE

Victor Casamayor Pujol
Universitat Pompeu Fabra
Barcelona, Spain
victor.casamayor@upf.edu

Boris Sedlak
TU Wien, Distributed Systems Group
Vienna, Austria

Tommaso Salvatori
VERSES AI
Los Angeles, USA

Karl Friston
University College London
London, UK

Schahram Dustdar
TU Wien, Distributed Systems Group Vienna, Austria
Universitat Pompeu Fabra Barcelona, Spain

ABSTRACT

The Computing Continuum (CC) is an emerging Internet-based computing paradigm that spans from local Internet of Things (IoT) sensors and constrained edge devices to large-scale cloud data centers. Its goal is to orchestrate a vast array of diverse and distributed computing resources to support the next generation of Internet-based applications. However, the distributed, heterogeneous, and dynamic nature of CC platforms demands distributed intelligence for adaptive and resilient service management. This article introduces a distributed stream processing pipeline as a CC use case, where each service is managed by an Active Inference (AIF) agent. These agents collaborate to fulfill service needs specified by *SLOiDs*, a term we introduce to denote Service Level Objectives that are aware of its deployed devices, meaning that non-functional requirements must consider the characteristics of the hosting device. We demonstrate how AIF agents can be modeled and deployed alongside distributed services to manage them autonomously. Our experiments show that AIF agents achieve over 90% SLOiD fulfillment when using tested transition models, and around 80% when learning the models during deployment. We compare their performance to a multi-agent reinforcement learning (MARL) algorithm, finding that while both approaches yield similar results, MARL requires extensive training, whereas AIF agents can operate effectively from the start. Additionally, we evaluate the behavior of AIF agents in offloading scenarios, observing a strong capacity for adaptation. Finally, we outline key research directions to advance AIF integration in CC platforms.

***K*eywords** Compute continuum, Active Inference, Service Level Objective, Multi-agent

1 Introduction

The next generation of Internet-based applications is poised to change how we live as a society: Autonomous driving will revolutionize urban transportation, demanding ultra-low latency to enable real-time decision-making [Lin et al.(2018)]. E-Health applications will allow for remote and detailed patient care, but will also require local data processing to protect patient privacy [Makina et al.(2024)]. Fleets of robots will autonomously clean and maintain city streets, ensuring efficient urban management [Zahidi et al.(2024)]. Resource usage, such as electricity and water [Shahra et al.(2024)], will be optimized through advanced distribution systems, and AR/VR technologies will reshape how we interact with people and objects, necessitating rapid processing of large data streams [Sukhmani et al.(2019)]. All these applications share the need for near-real-time computations while processing large amounts of data, which the Cloud alone cannot provide due to transmission latency [Hu et al.(2020)].

The Computing Continuum (CC) is considered by the community as the emerging platform with the potential capabilities to bring the required performance to these new applications [Beckman et al.(2020), Dustdar et al.(2023),

Nardelli et al.(2024)]. Integral to this continuum are its inherent heterogeneity and dynamism, fundamental aspects that must be addressed when developing effective management methods [Casamayor Pujol et al.(2023)]. These characteristics impact multiple facets of the system. For instance, heterogeneity is present in the types of computing and networking units, data modalities, system providers, and the variety of services within applications. Device heterogeneity, for example, hinders centralized auto-scalers because adaptations must be tailored to each device's capacity and characteristics [Zhang et al.(2019)]. Similarly, managing multiple system providers requires agreements and methodologies that go beyond current Cloud solutions, which typically rely on a single provider handling the entire infrastructure [Casamayor Pujol et al.(2024)]. Dynamism, on the other hand, affects devices, networking capacities, user demands, and system costs [Baek et al.(2020)]. Devices may have fluctuating capacities, making fast re-deployment or computation offloading necessary [Hu et al.(2023)]. Furthermore, dynamic costs and the involvement of multiple stakeholders require transparent rules to ensure fair and efficient application deployment and execution.

As a unified computing fabric that integrates all current computational tiers (i.e., IoT, Edge, Fog, and Cloud), the CC paradigm holds great potential to meet application requirements by harnessing the strengths of each tier [Beckman et al.(2020)]. The CC achieves this by placing the services' elasticity strategies directly where they are most needed, i.e., where services execute. Elasticity strategies refer to the ability of software services to adjust their configuration in response to changing conditions to maintain performance [Fürst et al.(2018)]. In Cloud environments, a common elasticity strategy is scaling the number of service replicas. In the Computing Continuum, however, strategies like offloading services to other devices or adapting the quality of data for processing are more common and effective [Manaouil and Lebre(2020)]. Local adaptation of services becomes essential to ensure the necessary performance for these applications, given the inherent heterogeneity and dynamism of the CC. This shift presents an exciting opportunity to move beyond the current Cloud-inspired model, paving context sensitivity for more advanced adaptation techniques.

Local adaptation implies that services must be able to monitor their status —what we refer to as Service Level Objectives in Device (SLOiD)— as well as their surrounding environment, which includes other related services, the hosting device, and even user behavior patterns. They need to analyze this information and then ask themselves: *Is an elasticity strategy necessary to improve current SLOiD fulfillment?* If so, *which one?* Based on this analysis, services must then adapt in real-time to meet current needs. This process requires services to continuously collect and interpret data to infer both the system's and the environment's state. These real-time decisions are crucial for maintaining performance but pose a complex planning challenge, as they must consider multiple factors and adjust actions on-the-fly. To address this challenge, we leverage Active Inference (AIF) [Friston et al.(2016)], a probabilistic agent-based approach that enables processing services to continuously adapt to a dynamic environment. From a technical perspective, AIF is particularly apt for the multiple and federated constraint problem furnished by SLOiDs; AIF inherits from the free energy principle, under which self-organisation is specified in terms of minimising surprisal or self-information. In AIF, this surprise is specified in terms of the negative log probability of occupying a preferred or characteristic state. Crucially, this means the objective function in AIF are functionals of probability distributions over outcomes — as opposed to simple functions of outcomes per se, such as reward, utility or cost-to-go. This means that the specification of constraints is naturally accommodated in terms of a probability distribution over all states or outcomes a system can experience. Furthermore, because free energy is an extensive quantity, if all the agents in a distributed architecture minimise their (expected) free energy, then the joint free energy of the ensemble is also minimised. This licences local (planning as) inference. In what follows, we leverage these two fundaments of AIF in the context of CC.

This article presents an AIF multi-agent system for managing a pipeline of distributed services. Each service agent is modeled as a partially observable Markov decision process (POMDP), which continuously evaluates its state and selects self-adaptive actions in real-time to optimize performance. To the best of our knowledge, this is the first attempt to use AIF and POMDPs in distributed processing pipelines of the CC. Hence, we present a detailed description on how we leverage AIF to encourage others researchers to test this novel approach. In addition, we discuss a methodology for integrating these agents into CC applications and provide a research roadmap outlining the requisite developments to make this technology fully actionable for real-world environments.

The rest of this article is organized as follows. We present the background in Section 2 needed to follow the article, specifically we discuss SLOiDs, POMDPs and AIF. Then, in Section 3 we introduce the overall vision for application management in the CC based on the key idea of distributing intelligence throughout the platform. Section 4 presents the methodology used to leverage AIF agents for a CC use case. The results of the experiments are presented in Section 5, which are used in Section 6 as a starting point to describe future developments that need to be addressed to use AIF in the CC. Afterwards, in Section 7 we present related work and summarize our findings in Section 8.

2 Background

In this section, we discuss the three concepts that are fundamental to the contributions made in this paper. First, we present SLOiDs as prevalent mechanisms for ensuring high-level service requirements. To ensure SLOiDs, we highlight how AIF, an emerging framework from neuroscience, can support the necessary resilient decision-making. In more detail, we introduce how these AIF-driven decision-making processes are modeled and optimized during runtime using POMDPs.

2.1 Service Level Objectives in Devices – SLOiDs

In Cloud computing, the promised service quality between infrastructure provider and application developer is specified within a Service Level Agreement (SLA). There, both entities agree on specific service-related measurements (Service Level Indicators - SLIs) that must behave in a specified manner (Service Level Objectives - SLOs), e.g., the CPU utilization of a computing node must stay under 80% [Beltran(2016)]. Then, the number of *good events*, e.g., number of measurements where the CPU stayed under 80%, is divided by the number of *valid events*, e.g., number of measurements considered, for a determined period of time [Beyer et al.(2016)]. For a Cloud business model, this value provides a notion of reliability of the service, which is used to define penalties (e.g., monetary compensations) for the infrastructure provider in case SLOs are violated. Indeed, the infrastructure provider will use elasticity strategies, e.g., scaling system resources up, when the SLO is not being fulfilled to reestablish the system's desired behavior [Dustdar et al.(2011)].

SLO Fulfillment: A probabilistic estimate of the system's performance which we aim to optimize. It is defined as the fraction of *good events* divided by the *valid events*, computed over a given period of time.

Applications distributed over the CC pose a variety of requirements (e.g., latency and/or quality) that define how each device and component should operate. However, contrarily to traditional Cloud computing, enforcing SLOs in the CC requires orchestration over numerous heterogeneous devices [Nardelli et al.(2024)]. To capture the complexity this adds, we extend the SLO definition and call them Service Level Objectives in Devices. In that regard, the key aspects that characterize SLOiDs, also in relation to regular SLOs, are as follows:

- Hardware heterogeneity directly dictates SLOiDs feasibility. Unlike the abstracted and relatively homogeneous resources in traditional Cloud environments, the CC is characterized by hardware diversity. SLOiDs cannot be practically defined without considering the specific capabilities and limitations (e.g., CPU power, memory availability, energy constraints, network bandwidth) of the target deployment devices. For instance, a latency SLO (e.g., response time below 50ms) achievable on a powerful Edge server is impracticable on a resource-constrained IoT sensor. Therefore, SLOiDs must be tailored to specific device classes. Ignoring this leads to perpetually unachievable objectives on lower-end devices or over-provisioning on higher-end devices.
- Scarce resources necessitate decentralized SLOiDs evaluation. The limited processing power, memory, and network bandwidth of CC devices make centralized, real-time monitoring, and evaluation of SLOiDs across the entire CC impractical or impossible. Constantly streaming detailed metrics from numerous constrained devices would overwhelm both the devices themselves (consuming scarce resources) and the network (causing congestion and increasing latency). Consequently, SLOiDs evaluation must be performed locally on the device or at a nearby edge node. This hardware-imposed limitation forces SLOiDs definitions to rely on metrics that can be efficiently computed and evaluated locally, potentially sacrificing global consistency or fine-grained observability for the sake of feasibility.
- Constrained hardware necessitates incorporating SLOiDs elasticity trade-offs. On many CC devices, particularly at the Edge and IoT layers, traditional cloud elasticity mechanisms like rapid scaling up/down or scaling out instances are often not viable due to strict hardware resource limits, power constraints, or physical deployment realities. To ensure service objectives, elasticity must involve adjustments in quality (e.g., reducing data fidelity, frame rate) or performance (e.g., accepting higher latency). Therefore, SLOiDs and elasticity strategies for services deployed on such hardware must be defined to anticipate and manage these trade-offs. This could involve allowing graceful degradation under certain conditions, explicitly acknowledging that the limitations of the hosting device may prevent consistently meeting a single, rigid target.

In summary, the heterogeneity and resource constraints of CC hardware are not just complicating factors; they are fundamental characteristics that dictate:

1. What types of SLOiDs are best suited for each part of the continuum.

2. How SLOiDs must be monitored and evaluated.
3. How SLOiDs must be defined to incorporate necessary trade-offs.

In the context of this work, fulfilling SLOiDs maintains the system in homeostasis, or *equilibrium* [Sedlak et al.(2024a)], as we called it in previous works. This is a preferred steady state that spans across numerous devices and applications; in particular, SLOiDs can create hierarchies where the fulfillment is dependent on multiple lower-level SLOiDs or processes [Casamayor Pujol et al.(2024)].

System Equilibrium: A computing system is in equilibrium if and only if all its SLOiDs are fulfilled.

Otherwise, if any SLOiD is violated, the system has lost its equilibrium; through controlled usage of elasticity strategies, it is possible to return the system to its equilibrium state. For instance, imagine a service under high load that violates its latency SLOiD, an elasticity strategy might throttle incoming requests or deploy an additional instance to reduce load, thereby returning the service to its equilibrium state. Failing to intervene risks that the service breaks beyond any possibility to returning to an equilibrium state. Furthermore, the concept of equilibrium can be complex. A system might possess multiple stable equilibrium states, each with different performance or efficiency characteristics. Elasticity strategies could potentially be used not just for recovery, but also to explore and transition between these states to optimize system operation.

2.2 Partially Observable Markov Decision Processes – POMDPs

A Markov-decision-process (MDP) is a time-discrete control process in which actions lead to partially random outcomes due to the stochastic nature of the system. In general, it models the transition probability of a new state given the current state of the system and the action of the agent. POMDPs generalize this to systems where the agent cannot observe all states of the system directly, but only the generated outcomes, or observations. As a simple example, consider an action that reduces the number of CPU cores available to a process; while this will affect the performance of the process, the extent of the impact is uncertain, as it may depend on other variables that are not directly observable. To address this kind of uncertainty, in our AIF agents for CC systems we will always specify the POMDP governing the service.

MDPs are time-discrete control processes in which actions lead to partially random outcomes due to the stochastic nature of the system.

In general, MDPs model the transition probability to a new state given the system's current state and an agent's action; hence, MDPs satisfy the Markov property, as counts for POMDPs.

In the context of AIF, a POMDP is defined by a 7-tuple (S, U, **B**, O, **A**, C, D), where:

- S is the set of states for the agent
- U is a discrete set of actions
- **B** are the state transition model
- O is the set of observations
- **A** are the conditional observation probabilities
- C is the set of preferred outcomes
- D is the belief over the initial agent's state.

For many challenging problems, representing the state S as a single, atomic label is inadequate. Real-world situations often involve multiple interacting factors that influence the system's evolution. For instance, whether a computation finishes on time might depend on both its computational complexity and the current availability of system resources. To handle such intricacies, we often employ a factored representation. In this approach, the overall state space S is defined as the Cartesian product of multiple state factors or modalities: $S = S_1 \times S_2 \times \dots S_N$. Thus, a specific state s is defined as tuple $s = (s_1, s_2, \dots, s_N)$, where each element s_i represents the value of the i-th factor (e.g., resource level, computation complexity) within its respective domain S_i. This factored structure naturally leads to representations of the transition (**B**) and observation (**A**) matrices that capture the interplay between these factors, often exploiting conditional independencies for a more compact and interpretable model.

The state transition models (B) They define the dynamics of the system, modelling $p(s_{t+1}|s_t, u_t)$, which is the probability of transitioning to state s_{t+1} given the current state s_t and action u_t. In the simple case with a single state

modality ($N = 1$), **B** can be viewed as a 3-tensor of shape $(|S|, |S|, |U|)$, where $|S|$ is the number of states and $|U|$ is the number of actions. When the state is factored into N modalities, $s_t = (s_{t,1}, \ldots, s_{t,N})$, representing the full transition function $p(s_{t+1}|s_t, u_t)$ as a single high-dimensional tensor becomes impractical both computationally and for descriptive purposes. The size of such a tensor would grow exponentially with the number of modalities N. To manage this complexity, we utilize the factored nature of the state space to represent the transition dynamics **B** in a structured and compact manner [Boutilier et al.(1999)]. This is commonly achieved using a *factored representation*, often based on a Dynamic Bayesian Network (DBN). The underlying assumption is that the state of a single modality at the next time step, $s_{t+1,i}$, typically depends only on a limited subset of state modalities at the current time step t (denoted as its parents, $Pa(s_i) \subseteq \{s_{t,1}, \ldots, s_{t,N}\}$) and the action u_t. Therefore, instead of defining one large tensor **B**, the transition model is specified by defining, for each modality i, its Conditional Probability Table (CPT): $P(s_{t+1,i}|Pa(s_i), u_t)$. These CPTs describe the local dependencies and how each modality evolves based on its specific influencing factors. These dependencies can be informed by an analysis of the system's behavior, such as the interdependencies illustrated in Figure 3. A precise definition of these CPTs requires identifying the relevant parent variables $Pa(s_i)$ for each s_i and quantifying their influence. Fortunately, the AIF framework allows for learning these parameters from interactions. In this work, we explore both expert-based specification and AIF-learned approaches for constructing the parameters of the factored transition model (**B**), assessing their respective challenges, benefits, and limitations. Notice that in Section 4.1, where the POMDPs for all agents are defined, we will specify the DBNs for each factor and the specific rules detailing the CPTs can be found at the Appendix.

The conditional observation probabilities (A) They define the likelihood of receiving a specific observation o_t given the system is in state s_t. That is, they model the distribution $p(o_t|s_t)$. This likelihood function is crucial for the agent, as its goal is typically to invert this mapping via inference (e.g., using Bayes' theorem) to estimate the hidden state s_t based on the observed outcome o_t. In its matrix form, **A** typically has dimensions $(|O|, |S|)$, where $|O|$ is the number of possible observations and $|S|$ is the number of states. In this work, however, we make a simplifying assumption of *perfect sensing*, meaning the observations correspond directly to the underlying states ($o_t = s_t$). We justify this assumption based on two primary reasons pertinent to our application domain. First, for the system under study, we possess *full observability* over the variables constituting the state representation. For example, we do not need to infer internal states like GPU usage from indirect measurements; we can directly monitor the relevant system parameters. Second, the precision of sensor data obtainable from modern computing devices is generally high relative to the granularity of our state representation. Since we employ categorical variables with reasonably large bins for discretization, the inherent measurement precision typically does not necessitate a complex sensor noise model within our POMDP framework. Consequently, for the agents defined in Section 4.1, the observation likelihood **A** reduces to an identity mapping. In the context of our factored state space $s = (s_1, \ldots, s_N)$, this means that for each modality i, the observation o_i perfectly reveals the state s_i, making the observation function an identity matrix for each factor. It is important to acknowledge that this perfect sensing assumption should be relaxed when modeling more complex scenarios involving significant partial observability or latent variables that must be inferred indirectly. Examples include tasks requiring the inference of hidden user intentions or internal states of external systems (e.g., inferring the success of a remote machine learning service based on subsequent user interactions). For such problems, defining a non-identity **A** matrix becomes essential.

Preferred Outcomes (C) This refers to a subset of possible outcomes that the agent aims to observe as a result of its actions. In our case, the fulfillment of SLOiDs is clearly a preferred observation for the agents. Note that preferred outcomes are not represented as probability distributions, but rather as log-probabilities. The preferred outcomes can be adjusted as the agent and the environment evolve, showing that preferences can change according to the current states.

Initial state (D) This component encodes the agent's belief over its initial state for each state modality. In this work, the initial state is randomized, but it can be specified to enforce particular scenarios if needed.

2.3 Active Inference – AIF

AIF is a corollary of the Free Energy Principle (FEP), that describes how agents resolve uncertainty in their understanding of the world by minimizing their variational free energy [Parr et al.(2022)]. More generally, the FEP applies to all kinds of adaptive agents that learn accurate world models, also called *generative models*, because they allow agents to understand and interpret observations *generated* by unobservable latent variables. Hence, agents are able to minimize the discrepancies between their generative model and the actual generative process that governs the environment [Friston(2013)]. To minimize their free energy, agents take epistemic actions that allow them to resolve uncertainty, but they also change the environment to move it into a state that fulfills its internal preferences, achieving homeostasis. This describes the exploration-exploitation tradeoff inherent

to agent-based systems. For more details about the FEP, the interested reader can check the following references [Friston et al.(2016), Kirchhoff et al.(2018), Friston et al.(2009), Friston et al.(2024b)].

In AIF, agents compute the expected free energy (EFE) for different policies (π), which are sequences of actions considered over various planning horizons. This allows agents to simulate their plan by computing the EFE for each action in the policy and selecting the one that minimizes EFE over the entire policy. In that regard, the policy length (*pl*) represents the number of time steps ahead that the agent simulates when evaluating policies. A longer *pl* allows the agent to consider more downstream consequences of actions when minimizing the EFE over the entire policy horizon. This means the optimal policy selected might involve an initial action that is not myopically optimal but enables lower EFE overall. However, increasing *pl* significantly increases the computational resources required for planning. The EFE can be broken down into two main components:

$$\text{EFE} = -\overbrace{\mathbb{E}_{Q(o|\pi)}[\ln P(o|C)]}^{\text{Pragmatic Value}} - \overbrace{\mathbb{E}_{Q(o,s|\pi)} D_{\mathrm{KL}}\left[Q(s|o) \,\|\, Q(s)\right]}^{\text{Information Gain}} \tag{1}$$

Here, the pragmatic value (*pv*) reflects the expected log probability, given the approximated world model (or, the posterior distribution) Q, of obtaining a desirable observation o under a specific policy π. The information gain (*ig*) represents the expected reduction in uncertainty (that is, the expected model improvement), expressed as the expected Kullback-Leibler divergence (D_{KL}) between the prior belief about the world state s, and the posterior belief computed after obtaining a new observation. By balancing these factors, a policy that minimizes EFE not only achieves desired outcomes in the short term, but also enhance the agent's understanding of the environment, leading to improved decision-making in the longer term. Hence, AIF ensures an accurate model for decision-making, allowing agents to persist over time [Palacios et al.(2020)]. Specifically for CC systems, this ensures long-term SLOiD fulfillment. This capability, closely related to *lifelong learning*, allows AIF agents to make decisions under uncertainty or dynamically changing environments. In particular, continuous exploration would foster alternative ways of scaling a service, and thus, increases its resilience.

3 Vision

Large scale computing systems, such as the CC, require mechanisms that underwrite each component's requirements in a decentralized manner. In this context, we want to highlight the term *distributed intelligence*, as it encompasses numerous concepts that are essential for achieving an equilibrium in distributed computing systems.

3.1 Distributed Intelligence

Contrarily to central Cloud services, the logic in CC systems is distributed over a widespread computing infrastructure. Figure 1 gives an intuition of a CC architecture that comprises multiple tiers, such as the IoT, and multiple computing layers from Edge, over Fog, up to the Cloud. As data travels from the IoT towards larger data centers, devices along the streaming pipeline can process the data through a network of microservices. Each processing service is supervised by a dedicated AIF agent that is responsible for (1) continuously evaluating SLOiD fulfillment and (2) taking actions whenever SLOiDs are violated, e.g., scaling the resources or quality of services. For taking actions, each agent uses a generative model that allows it to interpret the processing environment. Further, AIF agents (3) communicate and collaborate among themselves to exchange information (e.g., learned models or context) or actual workloads.

3.2 Problem description

Consider an application deployed within the CC that consists of three services organized in a pipeline. Each service performs a specific task, but they are interdependent, forming a sequential workflow. Using AIF agents, we will show how these services can autonomously adapt in a decentralized manner while cooperating to help each other achieve their objectives.

The use case consists of three services, which we refer to as the *Producer*, *Worker*, and *Consumer*, according to their position in the processing pipeline. Consider that the three services can form a chain in a smart city use case, where each service is hosted on different devices distributed across a city; this is also depicted in Figure 2:

The *Producer* service is attached to a video camera, interacting with the camera to prepare and send batches of images to a downstream service for further processing. The *Worker* service, deployed on an Edge server, performs tasks such as face detection and blurring to preserve pedestrian anonymity. Once the images are processed, they are sent to the final user of the image stream. The *Consumer* service runs on a smartphone or in an autonomous vehicle for traffic analysis, ensuring that the client's quality of experience (QoE) requirements are met to maintain user satisfaction with the overall application.

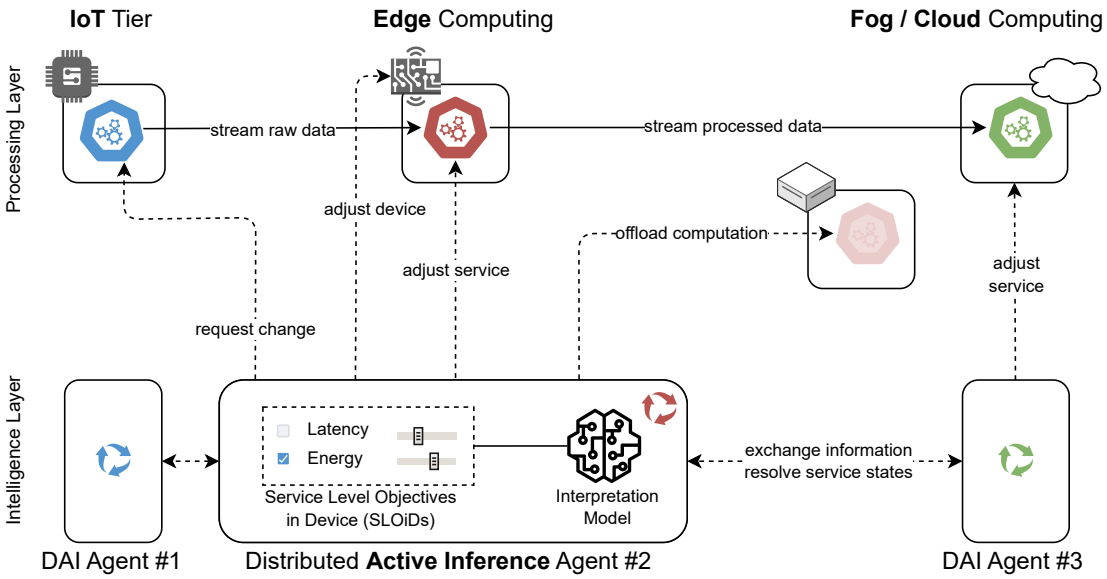


Figure 1: High-level vision of distributed intelligence in computing continuum systems; each processing service is supervised by a distributed AIF agent that evaluates on-device requirements, communicates with other AIF agents to exchange information and resolve states of other components, and takes action whenever SLOiDs are violated to restore the system equilibrium.

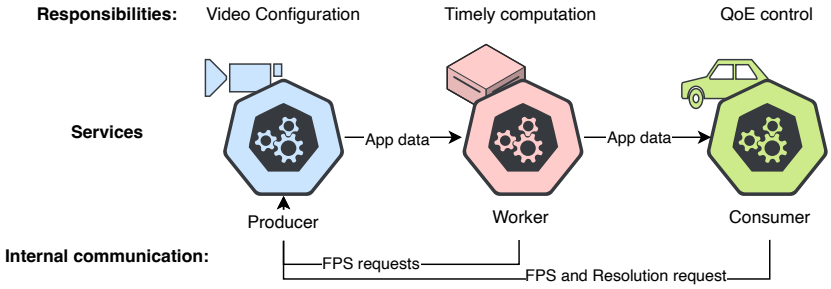


Figure 2: Smart city service pipeline from data producer to consumer. The *Producer* provides a video stream and adapts its configuration according to the needs of the *Worker* or *Consumer*. The *Worker* performs latency-sensitive processing, while the *Consumer* ensures QoE by influencing the configuration of upstream components.

Each service must fulfill a different set of SLOiDs and have different actions available. The *Consumer* SLOiDs are focused on controlling the user's Quality of Experience (QoE). Three are defined: the first, named **Success**, ensures that the service has correctly fulfilled its goal, e.g., the correct application of a privacy filter to each video frame. The second, **Smoothness**, measures the quality of the video stream by assessing the smoothness of consecutive frames. The third SLOiD, **C-consumption**, monitors the energy consumption of the service, including the extra cost needed to communicate with other services. The *Consumer* service has a single action available (*Toggle_comm*), which consists of enabling or disabling the communication channel with the *Producer* service in order to request a change on the video stream configuration, i.e., changing the frames per second or the resolution.

The *Worker* processes the video stream as a batch of images received within a specified deadline. The SLOiD **Latency**, ensures that the processing time for each batch remains below a given threshold. Like the *Consumer*, the *Worker* also has a **W-consumption** SLOiD, which tracks its energy consumption and the overhead associated with communicating with other services. Note that in both cases, communication refers to non-functional data exchange that improves control over SLOiDs. The *Worker* has two actions available: it can switch on and off its GPU in order to accelerate the images processing (*Switch_GPU*), notice that this is only possible when the deployment host has an available GPU. Second, it can enable or disable the communication channel (*Toggle_comm*) with the *Producer* service to request a change on FPS configuration of the video stream.

The *Producer* records the video stream with specific configurations and sends the images to the *Worker*. As the service with direct control over the recording settings, the *Producer* 's non-functional objective is to adjust the video configuration to satisfy both the *Worker* and the *Consumer*. Therefore, its three SLOiDs are **Worker satisfaction with FPS** (**WF**), **Consumer satisfaction with FPS** (**CF**), and **Consumer satisfaction with Resolution** (**CR**). When the *Producer* does not receive any request from the *Worker* or the *Consumer* or the request is not to change anything, the SLOiDs are considered fulfilled. The *Producer* has 2 possible actions which consist of changing the resolution (*Change_resolution*) and the FPS (*Change_FPS*) of its attached camera.

Table 1: Summary of the defined SLOiDs for the three AIF agents

Producer		*Worker*		*Consumer*	
Description		*Description*		*Description*	
Worker satisfaction with FPS	**WF**	Processing time	**Latency**	Correct application of the privacy filter	**Success**
Consumer satisfaction with FPS	**CF**	Energy consumption of the service	**W-consumption**	Quality of the final video stream	**Smoothness**
Consumer satisfaction with resolution	**CR**			Energy consumption of the service	**C-consumption**

4 Methodology

In this article, we demonstrate how intelligence can be distributed across CC services by modeling agents using AIF, enabling them to manage services autonomously and collaboratively, an essential capability for decentralized application management in the CC. Now, we formally define the AIF agents as POMDPs, describe the tools and simulation setup, and conclude with a description of the experiments.

4.1 POMDPs definition

4.1.1 *Producer*'s POMDP model

State modalities (S) The *Producer*'s state (S) contains five modalities. Three of them are SLOiDs: **WF**, **CF**, and **CR**, as in Table 1. The other two state modalities are *FPS* and *Resolution*. The **WF**, **CF**, and **CR** modalities can each take three state values: *Increase*, *Stay*, and *Decrease*. The *Stay* value is assumed when no message is received from either the *Consumer* or the *Worker*, as if they are satisfied with the current value. The *FPS* modality can take the values $\{12, 16, 20, 26, 30\}$, and the *Resolution* modality can take the values $\{120p, 180p, 240p, 360p, 480p, 720p\}$. Figure 3 illustrates the modalities and actions of the *Producer*, and how they influence the state modalities at the next time step. This can be interpreted as a Dynamic Bayesian Network (DBN) representing the evolution of the *Producer* 's state modalities, with the assumption that the relationships between variables remain fixed at each time step.

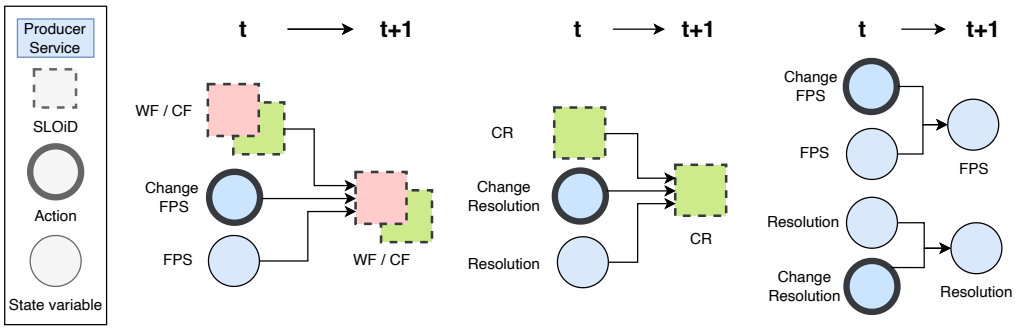


Figure 3: *Producer* 's DBN. The model includes five modalities, two of which behave identically and are therefore grouped together in the figure. The colors correspond to the services each modality is associated with.

Actions (U) The *Producer* 's action set (U) consists of two actions, each with three possible values.

1. *Change_FPS*: *Increase*, *Stay*, and *Decrease* the *FPS* current value.
2. *Change_resolution*: *Increase*, *Stay*, and *Decrease* the *Resolution* current value.

Consider that to limit the exponential growth of possible actions, both *Increase* and *Decrease* modify the current value by one step, e.g., increasing resolution from $120p$ to $180p$.

State Transition Model (B) The dependencies defining the transition probabilities **B** for each state modality, given other system states at time t and actions u_t, are illustrated in Figure 3. We define the Conditional Probability Tables (CPTs) for each modality using the rules described in Appendix A. For the alternative case where the AIF agent learns the transition parameters, the CPTs are initialized with uniform probability distributions instead of the deterministic rules. However, the underlying DBN structure defining the dependencies (as shown in Figure 3) remains identical in both the expert-defined and learned models. This applies for the 3 POMDP models described, i.e., the *Producer*, *Worker*, and *Consumer*.

Observations (O) The set of observations represents the possible outputs of the external environment, produced after an action $u \in U$ has been taken. As explained earlier, in this work we are assuming a perfect sensing. That is, an identity mapping between states and observations. To this end, the set of observations (O) for the agent is equivalent to the set of states (S). Consequently the likelihood mapping A is defined as a set of identity matrices, one per modality. This applies for the 3 POMDP models described.

Preferred Ouctomes (C) In this work, we consider the *Producer* to have no preferences over the *FPS* and *Resolution* sets. The preferred outcomes of the requests from the *Worker* and the *Consumer* are to receive a *Stay*, meaning their configuration requests are satisfied. More in detail, the vectors are initially defined as follows: $C^{\mathbf{WF}} = \{0.25, 1.5, 0.25\}$; $C^{\mathbf{CF}} = \{0.5, 3.0, 0.5\}$; and $C^{\mathbf{CR}} = \{0.5, 3.0, 0.5\}$. It is worth mentioning that the requests from the *Consumer* have a higher maximum value than the ones of the *Worker*, to indicate the ultimate goal is achieving satisfaction at the end of the pipeline, and considering that the *Worker* might have other capabilities to satisfy its goals.

Initial state (D) We consider the initial state to be unknown, its values are uniformly distributed adding up to 1. This applies for the 3 POMDP models described.

4.1.2 *Worker*'s POMDP model

State modalities (S) The *Worker* has six state (S) modalities. Two of them are SLOiDs: **Latency** and **W-consumption**, as in Table 1. The others are state variables: *Execution Time*, *FPS*, *Share Information*, and *GPU*, all can be seen in Figure 4. Each modality can take specific values based on the system's behavior. **Latency** is a boolean modality: it takes the value *True* when the computation is completed within the deadline, and *False* when the computation exceeds the deadline. *Execution Time* is a categorical modality representing the time required to complete the computational task. It can take the values {*LOW, MID-LOW, MID, MID-HIGH, HIGH*}, corresponding to the following durations in milliseconds: $(0\,,\,15], (15\,,\,30], (30\,,\,45], (45\,,\,60]$, and $(60\,,\,\infty)$. The *FPS* state modality is defined as for the *Producer*. **W-consumption** is a state modality that reflects the energy consumption of the device, considering GPU usage and the activation of the upstream communication channel. The levels are *LOW*, *MID*, and *HIGH*, corresponding to an overall consumption below 7W, between 7–8W, and above 8W, respectively. *Share Information* is a boolean state modality that is *True* when the upstream communication channel is active, allowing the *Worker* to communicate with the *Producer*, and *False* when it is disabled. *GPU* is also a boolean state modality: it is *True* when the GPU is on and *False* when it is off.

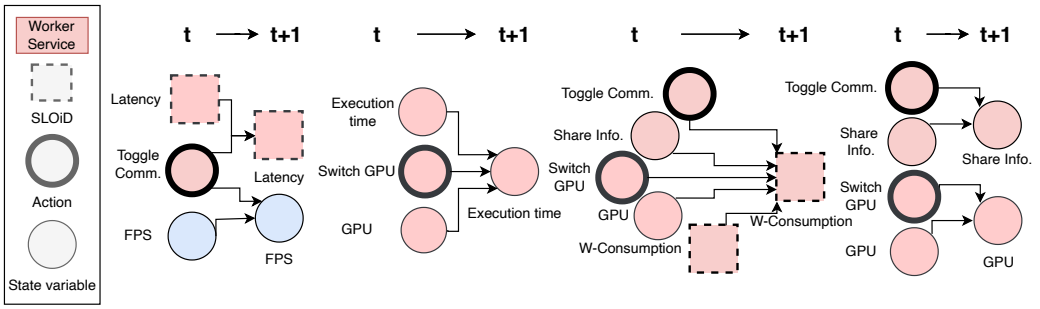


Figure 4: *Worker* 's DBN. The model includes six modalities, two of which are SLOiDs. The colors correspond to the services each modality is associated with.

Actions (U) The *Worker*'s action set (U) consist of two actions:

1. *Switch_GPU*: *Switch off* and *Switch on* modifies the GPU current state and *Stay* keeps the current state.
2. *Toggle_comm*: *Enable* and *Disable* the upstream communication channel.

State Transition Model (B) The dependencies defining the transition probabilities **B** for each state modality, given other system states at time t and actions u_t, are illustrated in Figure 4. We define the Conditional Probability Tables (CPTs) for each modality using the rules described in Appendix B.

Preferred Outcomes (C) There are no preferred outcomes for the *FPS*, the *Share Information*, or the *GPU*, hence they are initialized as a vector of zeros. The **Latency** preferred outcome is *True*, hence the vector is defined $C^{\textbf{Latency}} = \{0.1, 3\}$. The *Execution Time* prefers values that are *MID* or below, the vector is: $C^{\text{Exec. Time}} = \{3, 2.5, 2, 0.25, 0.1\}$. Finally, the **W-consumption** is expected to be *LOW* or *MID*, its vector is: $C^{\textbf{W-consumption}} = \{3, 2.5, 0, 5\}$

4.1.3 *Consumer*'s POMDP model

State modalities (S) The *Consumer*'s state (S) consists of six modalities. Three of them are SLOiDs: **Success**, **Smoothness**, and **C-consumption**, as in Table 1. The other three are system variables: *FPS*, *Resolution*, and *Share Information*, as can be seen in Figure 5. Each modality can take specific values based on the system's behavior. **Success** is a boolean modality: it takes the value *True* if the service has properly applied the privacy filter, and *False* otherwise. **Smoothness** is a categorical state modality representing the distance shift of a pixel between two consecutive frames. It can take the values (*SHORT, MID-SHORT, MID, MID-LONG, LONG*), which correspond to a pixel movement of $(0-25], (25, 50], (50, 75], (75, 100], (100, \infty)$ pixels. The **C-consumption** modality is defined as for the *Worker*, but without the influence of the GPU status, only considering the device's energy consumption and the activation of upstream communication. Both the *FPS* and the *Resolution* state modalities are defined similarly to those for the*Producer*. Finally, the *Share Information* is defined in the same way as for the *Worker*.

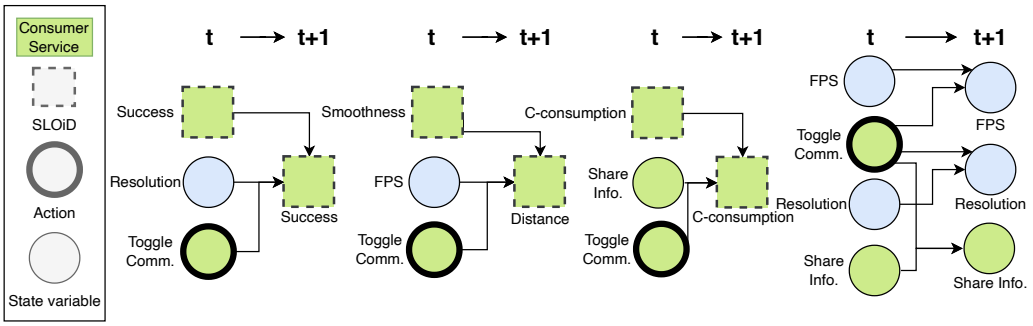


Figure 5: *Consumer* 's DBN. The model includes six modalities, three of which are SLOiDs. The colors correspond to the services each modality is associated with.

Actions (U) The *Consumer* action set (U) consist of:

1. *Toggle_comm*: *Enable* and *Disable* the upstream communication channel.

State Transition Model (B) The dependencies defining the transition model **B** for each state modality, given other system states at time t and actions u_t, are illustrated in Figure 5. We define the Conditional Probability Tables (CPTs) for each modality using the rules described in Appendix C.

Preferred Outcomes (C) There are no preferred outcomes for the *FPS*, *Resolution*, or *Share Information*, so their vectors are all zeros. The **Success** preferred outcome is *True*, hence the vector is as follows: $C^{\textbf{Success}} = \{0.25, 3\}$. The **Smoothness** preferred outcome is MID or lower, so the vector is: $C^{\textbf{Smoothness}} = \{3, 2.5, 2, 0.5, 0.1\}$. Finally, the **C-consumption** preferred outcome is MID or lower, so the vector is: $C^{\textbf{C-consumption}} = \{3, 2.5, 0.5\}$.

4.2 Tools and simulation setup

In this section we present the AIF library used, the dataset, the key elements of the simulation process.

4.2.1 Active inference with *pymdp*

All experiments in this article are conducted using *pymdp* [Heins et al.(2022)], which is a Python library that simplifies the implementation of AIF agents. *pymdp* provides a set of classes and methods for modeling, inference, learning, and

control: the core class, `Agent`, allows to define an AIF agent, whose methods provide the key functionalities needed for AIF agents, such as policy/action selection, state inference, and parameter learning.

It is important to note that the type of actions performed by the agents, as well as the complex relationships between state modalities and actions, are not commonly encountered in standard AIF scenarios. Therefore, the version of *pymdp* used for this work was based on a customized fork from GitHub[1], which is specifically adapted to handle these types of complex state transitions.

4.2.2 Dataset

The dataset are the traces of the processing environment of the pipeline of streaming services. The traces have been recorded using different hardware to account for the diversity of the CC. The dataset records various system parameters, including execution time, CPU utilization, memory usage, energy consumption, image resolution, frames per second (FPS), the success of the privacy model, the smoothness between consecutive frames, the type of device, and GPU usage in the described pipeline. So, it provides all the required data to rebuild the use case for this article. The dataset is publicly available on GitHub[2].

The use case described here is similar to that presented in [Sedlak et al.(2024a)], in the sense that it consists of multiple sequential processing services. However, whereas in [Sedlak et al.(2024a)] the AIF agent controls the entire system, in this paper, each service is associated with an autonomous agent, making the system both multi-agent and decentralized.

Further, the dataset was generated by independently adjusting key system parameters —image resolution, video frame rate, and GPU status— providing an independent set of behaviors for each parameter combination. This makes the dataset particularly well-suited for our simulation experiments, as it allows us to model independent behaviors and interactions between variables.

The dataset is used to simulate different operational scenarios for the pipeline, helping to validate the performance of the active inference agents under varying conditions. Since the dataset missed few combinations of resolution and FPS, some missing FPS values were approximated. Specifically, the missing FPS values were interpolated by averaging the neighboring values. The standard deviation of these neighboring values was also considered to ensure a realistic approximation of the missing data.

4.2.3 Simulation

Several simulation experiments are conducted to identify the gaps needed to leverage active inference in the context of Computing Continuum system, to make distributed intelligence a reality. Combining both disciplines (AIF and the CC) presented various challenges, and running experiments helped us to better understand how they can be integrated effectively. The code for these simulations can be found in the following GitHub Repo[3].

Each experiment was run for 200 time steps and repeated at least 10 times to account for the variations in initial conditions. These correspond to the initial values of the system's parameters: image *Resolution*, *FPS*, *GPU* status, *Worker* sharing information, and *Consumer* sharing information. Randomized initial configurations are used in every repetition ensuring that agents could start in diverse states, including possible equilibrium states where no immediate action is required.

The simulation process can be summarized as follows:

1. **Initialization.** At the beginning of each experiment, the agents' initial state and parameters are set randomly.
2. **Policy computation.** Each agent computes the best policy according to the active inference framework, i.e., a policy that minimizes the EFE.
3. **Action selection.** The agents selects the first action and applies it to the system. Consequently, the environment and the system changes. It is important to remark that agents can take all available actions at each time steps, meaning that the *Producer* can simultaneously change the resolution and the FPS of the camera.
4. **New Observation.** The environment generates a new observation for each agent by sampling from the dataset; this sample need not follow a temporal sequence but must correspond to the result of the chosen action.
5. **Infer state and learn parameters.** Based on these new observations, agents can infer their new state and learn the parameters of the transition model.

[1] https://github.com/ran-wei-verses/pymdp

[2] https://github.com/borissedlak/analysis/tree/main/data, last accessed on April 25th, 2025

[3] https://github.com/vikcas/AIF_for_CC

6. **Repetition.** At this point a new policy is computed, repeating the cycle for 200 times.

4.3 Evaluation

During the experiments, we evaluated two key aspects:

- SLOiDs fulfillment rate. The agent's ability fo meet its service goals. For visualization purposes, at each time step the cumulative SLOiD fulfillment rate is shown.
- Expected Free Energy. A measure used by AIF to assess their performance in terms of selecting the action that helps them improve their model while fulfilling the SLOiDs, see Equation (1).

Each agent has the following SLOiDs fulfillment criteria:

- *Producer*: **WF**, **CF**, and **CR** = { *Stay* }
- *Worker*: **W-consumption** $\leq$ { *MID* } and **Latency** = { *TRUE* }
- *Consumer*: **Success** = { *TRUE* }, **Smoothness** and **C-consumption** $\leq$ { *MID* }

4.3.1 Experiments

We conduct five experiments to assess the suitability of AIF agents for autonomous and distributed management of SLOiD fulfillment across the service pipeline.

SLOiD fulfillment This experiment evaluates how the three AIF agents fulfill their respective SLOiDs by selecting appropriate actions given each state of the environment. We fix the policy length to 3 and use the defined transition model. Results are presented in Section 5.1.

Learning the transition model This experiment assesses the impact of learning the transition model (**B**) on the agent's performance. We again set the policy length to 3, use the default learning parameters from the *pymdp* library, and initialize the transition model with **B** matrices, each containing a uniform distribution over transitions from any current state to the next. Results are presented in Section 5.2.

Comparison to multi-agent reinforcement learning This experiment uses the same environment than the 2 previous experiments, but the SLOiDs are controlled by 3 reinforcement learning-based agents. We use an algorithm from the state-off-the-art named Proximal Policy Optimization [Schulman et al.(2017)] and compare the results with the previously obtained as a baseline for the AIF agents. Results are presented in Section 5.3.

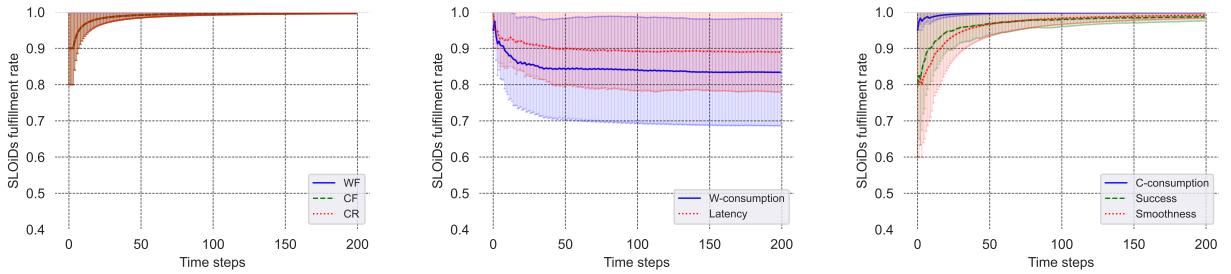
Heterogeneous hardware This experiment explores the response of the AIF agents to severe environment changes, mimicking as if they were redeployed in different hardware. To do so, after 75 time-steps into controlling the services in a specific device, they are change to a different one. We see both the dynamic adaptation of the system, as well as the need of adapting the SLOiDs to the pair service-hardware. Results are presented in Section 5.4

Computing cost This experiment analyzes the computational cost and the resulting performance of AIF agents. We compare the initial results obtained to those obtained with a shorter policy length. We have seen that given the same models, the policy length has a large effect on the computational cost. Results are presented in Section 5.5.

5 Results

5.1 SLOiD fulfillment

This experiment assesses the agents' ability to select appropriate actions based on expert-defined transition models. As shown in Figure 6, all agents stabilize their SLOiDs fulfillment rates around 50 steps. However, the *Worker* agent clearly struggles more to achieve high SLOiDs fulfillment rates compared to the others. This reduced performance is primarily due to the competing SLOiDs integrated into its model. Specifically, the **W-consumption** mandates switching off the GPU and communication channel, while the **Latency** requires them to be switched on if the proper setting of the camera is not selected. In addition to its lower maximum SLOiD fulfillment rate, the *Worker* agent also exhibits greater performance variance. This higher variance may stem from the *Producer* 's preference for prioritizing the satisfaction of the *Consumer* over that of the *Worker*. As a result, even if the *Worker* consistently selects optimal actions, the outcome may not meet expectations, depending on the requests from the *Consumer* to the *Producer*.



(a) *Producer* SLOiDs fulfillment rate for **WF**, **CF**, and **CR**.

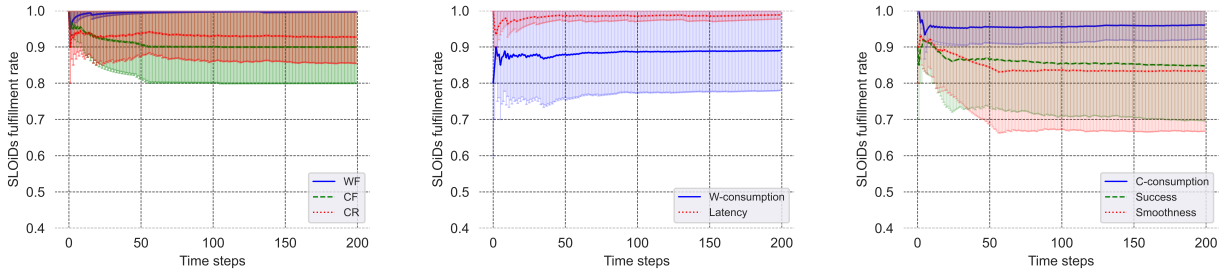
(b) *Worker* SLOiDs fulfillment rate for **Latency** and **W-consumption**.

(c) *Consumer* SLOiDs fulfillment for **Success**, **Smoothness**, and **C-consumption**.

Figure 6: SLOiDs fulfillment rate with no **B** parameter learning and policy length of 3.

5.2 Learning the transition model

Figure 7 shows the SLOiD fulfillment rate for each AIF agent in a variation of the previous experiment where agents must select actions to fulfill SLOiDs while simultaneously learning the transition model. In comparison with the results from the previous experiment (Figure 6), we observe that all agents stabilize their SLOiD fulfillment rates also around the 50 time steps. The overall fulfillment rate of the agents is lower than in the previous case. This indicates increased difficulty in finding performance-maximizing policies when the model is unknown and must be learned online. For example, agents *Producer* and *Consumer*, which reached nearly 100% fulfillment in the previous experiment, are around 80-90% in this learning scenario. An interesting observation is that in these learning trials, the *Worker* agent appears to prioritize the **Latency** over **W-consumption**, resulting in better fulfillment rates compared to its performance in the previous experiment with the expert model.



(a) *Producer* SLOiDs fulfillment rate for **WF**, **CF**, and **CR**.

(b) *Worker* SLOiDs fulfillment rate for **Latency** and **W-consumption**.

(c) *Consumer* SLOiDs fulfillment for **Success**, **Smoothness**, and **C-consumption**.

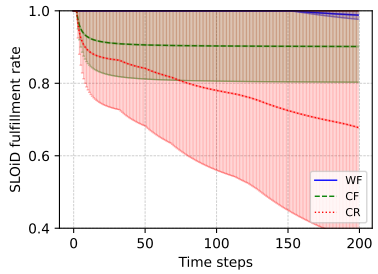
Figure 7: SLOiDs fulfillment rate with **B** parameter learning and policy length of 3.

5.3 Comparison to multi-agent reinforcement learning (MARL)

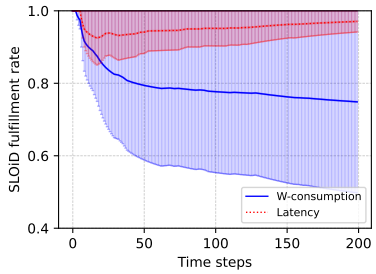
The same three agents were implemented within a multi-agent reinforcement learning (MARL) framework using RLlib [Liang et al.(2018), Wu et al.(2021)] and the PPO algorithm [Schulman et al.(2017)]. Consistent with the AIF experiments, we did not fine-tune any parameters, allowing us to assess each method's general capabilities in this scenario. We consider two distinct training regimes: minimal (400,000 time steps) and extensive (40 million time steps), mirroring the exploration of two scenarios with the AIF agents. Evaluation is performed over 200 time steps. The reward function of each agent is computed by assigning a reward for every SLOiD it fulfills. Therefore, actions that lead to fulfilling more SLOiDs result in higher rewards.

Figure 8 shows the SLOiD fulfillment rates for all the MARL agents. The top three plots (Figures 8a, 8b, and 8c) correspond to agents trained under the minimal training regime. In this case, their performance is clearly lower than that of the AIF agents. The *Producer* agent fails to stabilize consumer satisfaction with resolution, and the *Worker* agent shows a declining trend in fulfilling the **W-consumption** SLOiD. In contrast, the three plots at the bottom (Figures 8d, 8e, and 8f) correspond to the extensive training regime, where agents have acquired sufficient knowledge of the environment to consistently fulfill all their SLOiDs.

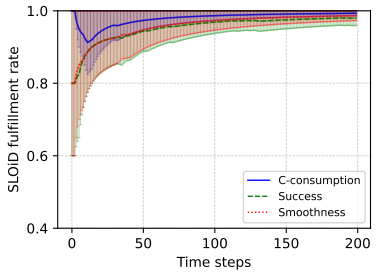
Although this article does not aim for a direct comparison between the two approaches, it is worth noting the significant data efficiency of the AIF agents. Specifically, the learning AIF agents achieved their results with less than 200 time steps of runtime data, a stark contrast to MARL agents which model the environment through neural networks, which typically require large amounts of data.



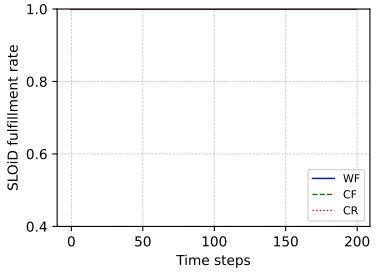
(a) Minimally trained MARL *Producer* agent. SLOiDs fulfillment rate for **WF**, **CF**, and **CR**.



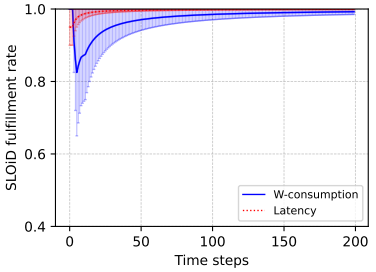
(b) Minimally trained MARL *Worker* agent. SLOiDs fulfillment rate for **Latency** and **W-consumption**.



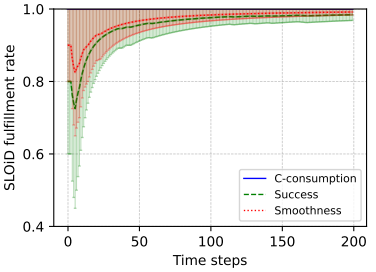
(c) Minimally trained MARL *Consumer* agent. SLOiDs fulfillment for **Success**, **Smoothness**, and **C-consumption**.



(d) Extensively trained MARL *Producer* agent. SLOiDs fulfillment rate for **WF**, **CF**, and **CR**.



(e) Extensively trained MARL *Worker* agent. SLOiDs fulfillment rate for **Latency** and **W-consumption**.



(f) Extensively trained MARL *Consumer* agent. SLOiDs fulfillment for **Success**, **Smoothness**, and **C-consumption**.

Figure 8: MARL simulation, showing the SLOiD fulfillment rates for the minimally- and well-trained agents

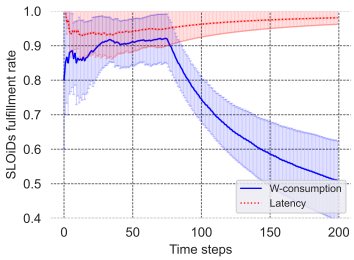
5.4 Heterogeneous hardware

Figure 9 illustrates the experimental results when the hosting environment of the AIF agents was deliberately and suddenly altered, simulating an offloading of their services to different hardware. Specifically, Figure 9a displays the performance of the AIF *Worker* agent while learning the transition model, whereas Figure 9c shows the same *Worker* agent operating with a known transition dynamics model. Prior to analyzing these results, it is crucial to note that the new host exhibits significantly higher energy consumption compared to the initial one. Consequently, fulfilling the **W-consumption** SLOiD becomes exceedingly challenging without redefining it. Bearing this in mind, we observe a clear degradation in the **W-consumption** SLOiD for both *Worker* agents. This degradation is more pronounced for the learning agent. However, the learning agent demonstrates a faster reduction in the variance of the **Latency** SLOiD, which can be attributed to its superior adaptability. Furthermore, Figure 9b presents the expected free energy (EFE) for the AIF *Worker* learning agent during the same experiment. It is evident that the agent initially reduces its EFE. However, the EFE suddenly increases due to the environmental change, after which the agent successfully resumes its EFE reduction. In contrast, Figure 9d depicts the EFE for the AIF *Worker* non-learning agent. This agent also experiences a change in its EFE, primarily characterized by a substantial reduction in its variance. This suggests a narrowed range of possible actions for the agent following the offloading process.

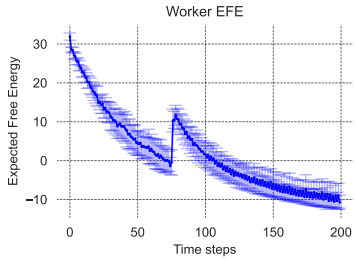
5.5 Computing cost

Two main factors influence the computational cost for AIF agents: (1) learning the transition model, and (2) the length of the planned policy. In this analysis, computational cost primarily refers to execution time.

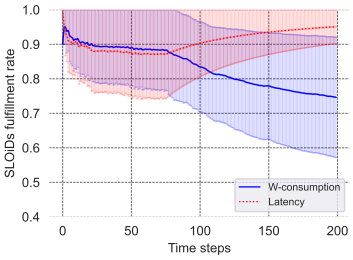
On average and in a Apple M3 chip with 18GB of RAM, a single time step in the environment takes approximately 80.55 seconds when the transition model must be learned for a policy length of 3. In contrast, when transitions are provided, the same step takes only 1.17 seconds on average. While these durations naturally depend on the computational capacity of the device, the ratio between them offers a more device-agnostic metric: learning the transition model



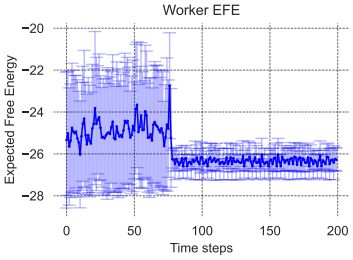
(a) *Worker* SLOiDs fulfillment while learning transition model and facing sudden environmental change.



(b) Expected Free Energy for the *Worker* learning the transition model.



(c) *Worker* SLOiDs fulfillment with known transition model and facing a sudden environmental change.

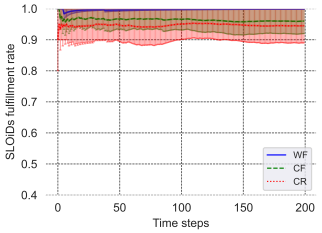


(d) Expected Free Energy for the *Worker* without learning the transition model.

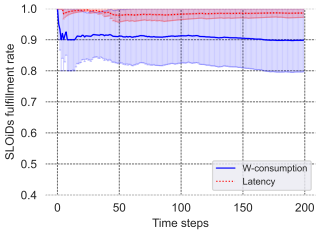
Figure 9: AIF *Worker* agent suffering an offloading to a new and different host after 75 time steps controlling its SLOiDs.

results in a slowdown of approximately $69.08\times$. Interestingly, this computational burden diminishes with shorter policy horizons due to the reduced combinatorial complexity. When repeating the experiment with a policy length of 1, the slowdown ratio is halved, the learning agent is approximately $35\times$ slower than its counterpart with provided transitions.

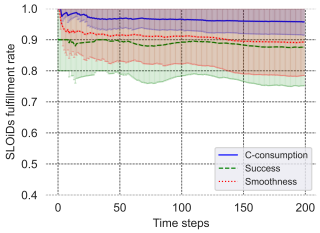
Given this computational overhead, it is important to evaluate agent performance under constrained policy lengths. To this end, we assess SLOiD fulfillment rates under the same setup used in previous evaluations. When agents had to learn the transitions with a policy length of 1, agents achieved fulfillment rates comparable to those with a policy length of 3, see Figure 10. Surprisingly, the *Producer* agent fulfillment rate (Figure 10b) was consistently higher for length-1 policies than for length-3, which was unexpected. This result may stem from the relatively small number of repetitions (10) and the stochastic nature of the environment, where identical actions in the same state can yield different outcomes. Additionally, most agent actions are boolean, and the environment exhibits limited temporal dependencies. As a reminder, environmental responses are based on the agent's action in the previous state, and new states are sampled to fulfill constraints without necessarily maintaining temporal continuity. Consequently, in this scenario, policy length may not be a critical factor in minimizing expected free energy (EFE)[4]. However, it is a key factor to consider in order to minimize the resource requirements of the agents.



(a) *Producer* SLOiDs fulfillment rate, they are **WF**, **CF**, and **CR**.



(b) *Worker* SLOiDs fulfillment rate, they are **Latency** and **W-consumption**.



(c) *Consumer* SLOiDs fulfillment rate, they are **Success**, **Smoothness**, and **C-consumption**.

Figure 10: SLOiDs fulfillment rate with **B** parameter learning and policy length of 1.

[4]Although longer policies generally achieve lower EFE values, this effect is partly an artifact of EFE computation: longer policies distribute cumulative EFE over more steps, which lowers the average per step.

6 Discussion

The results indicate that AIF agents effectively support distributed intelligence in the CC, enabling interdependent services to collaborate while maintaining consistent performance, quality, and cost. This work represents a first step toward realizing that vision. In what follows, we examine the gap between our current implementation and the broader potential of intelligence distribution using AIF in the CC. We organize the discussion into design-time and runtime considerations, while acknowledging that several aspects span both.

6.1 Design

6.1.1 AIF Agent Model

This work introduces an AIF agent model for supervising services based on predefined SLOiDs. Defining these SLOiDs is critical to ensure service compliance with performance expectations in its execution environment. This task is simpler when prior deployment knowledge exists; otherwise, changes in host devices can compromise SLOiDs fulfillment due to mismatched assumptions.

Modeling the AIF agent requires identifying dependencies between SLOiD variables and system variables. These dependencies—especially those involving inter-service interactions may remain hidden without cross-service analysis [Casamayor Pujol et al.(2024)]. Causal discovery methods like those by Mariani et al. [Mariani and Zambonelli(2024)] offer potential support, although typically post-deployment. In this study, we modeled service data as Bayesian Networks (BNs) and derived Markov Blankets as a prerequisite for both learned and predefined transition models [Sedlak et al.(2024a)].

Once dependencies are identified, they must be quantified, leading to two key design choices: (i) whether to model variables as discrete or continuous, and (ii) how to acquire the relationship parameters.

We chose discrete variables for computational tractability. This required determining variable granularity, number of classes and class thresholds, effectively coarse-graining the agent's model and limiting behavioral resolution. Continuous modeling, while more expressive, demands richer data and assumptions on variable distributions [Çatal et al.(2020)], and is computationally intensive.

Regarding quantification, we explored both deterministic (expert-based) and learned approaches (see subsections 5.1 and 5.2). Expert-based models are stable but rigid, and increasingly inadequate as system complexity grows. Learned models offer adaptability and autonomy, but incur high computational cost and demand longer convergence times.

6.1.2 Deployment

Although our experiments relied on simulations, deploying AIF agents alongside services within a real CC infrastructure is essential. This requires tight integration between services and their supervisory agents to enable real-time SLOiD adaptation or parameter adjustments. Such integration promotes a novel design paradigm in which services natively support adaptive supervision through new interfaces, enabling autonomous control and continuous optimization.

6.2 Runtime

At runtime, a core challenge is the combinatorial explosion of state-action pairs. To mitigate this, we used binary actions (e.g., *Switch off*, *Switch on*), and occasionally a neutral option (*Stay*), which reduced complexity but limited precision. This trade-off can hinder the agent's ability to reach target states efficiently when fine-grained control is needed. The effect worsens with longer policy lengths, making it critical to match action granularity and policy horizon to the environment's dynamics. We propose addressing this with hierarchical action-selection: AIF agents choose high-level actions, while a secondary controller handles precise execution [Pezzato et al.(2020), Parr and Friston(2018), Tschantz et al.(2022)]. Such hybrid strategies, as suggested in [Collis et al.(2024)], could improve adaptability and efficiency.

Flexibility is also vital in dynamic settings. Services may adjust preferences based on context or stakeholders. AIF agents accommodate this via direct updates to expected outcomes (C), adapting to runtime goals. Software updates, common in CC, may alter service behavior or capabilities. While minor behavior changes can be handled through continual learning, new capabilities may obsolete the agent's model. Detecting and adapting to such changes—or enabling agents to extend their models accordingly—remains an open challenge. This leads to the concept of model transfer, sharing models across similar services or devices could accelerate adaptation and support federated learning-

like knowledge aggregation. Future work should explore mechanisms for safe and effective model transfer between AIF agents.

7 Related work

In the context of this work, we identified two topics that have been addressed to some extent in existing work, namely adaptive mechanisms for the Computing Continuum, and distributed intelligence through Active Inference. The first topic targets the problem domain and the latter one the methodology applied. We present a concise description of related work and contrast it with our presented work to highlight the research gap we aim to fill.

7.1 Adaptive Mechanisms for the Computing Continuum

The CC is an emerging paradigm that has only been developed and extended over the last recent years of research [Dustdar et al.(2023)]; in particular, there remain multiple challenges in orchestrating such large-scale systems [Casamayor Pujol et al.(2023)], which require intelligent adaptation mechanisms. As such, Filinis et al. [Filinis et al.(2024)] present an intelligent auto-scaling agent for the CC, which scales the replicas of serverless functions that are deployed over a continuum. Closely related to that, Zafeiropoulos et al. [Zafeiropoulos et al.(2024)] provide a Reinforcement Learning (RL)-based auto-scaling environment for CC systems that explored the synergies between low- and high-level controlling entities to achieve globally optimal solutions. Their core motivation was to combine SLOs with dynamic processing requirements, which also applies to *Octopus*, a framework developed by Zhang et al. [Zhang et al.(2023)]. *Octopus* finds optimal service configurations in multi-tenant edge computing scenarios; for this, it predicts SLO fulfillment of two variables based on a deep neural network. To detect SLO violations of a scheduling task, Shubha et al. [Shubha and Shen(2023)] presented *AdaInf*, which can find SLO-fulfilling resource allocations between model training and inference; such mechanisms will prove essential to the CC.

While research on adaptive cloud mechanisms has made significant progress [Verma and Bala(2021)], the CC is still unexploited, but recent advances in Edge orchestration techniques promise to improve this. For example, through offloading techniques between mobile Edge devices and stationary computing resources, like Fog or Cloud. As such, Ma et al. [Ma et al.(2024)], Wu et al. [Wu et al.(2023)], and Spring et al. [Spring et al.(2025)] provide orchestration mechanisms that can be integrated into the CC. To make accurate predictions of how intent-based services in the CC will behave, Akbari et al. presented *iContinuum* [Akbari et al.(2024)], a simulation environment that can run experiments in Edge-to-Cloud scenarios.

Given the presented research, we conclude that research on the CC is still at its beginning, if not at an infant stage, with most works focusing on the potentials and challenges of the CC and comparably few empirically-evaluated solutions. As Edge computing advances, including the research on multiple distributed agents in Edge networks, orchestration mechanisms, as presented in this paper, will be heavily needed to ensure collaboration between multiple computational tiers.

7.2 Distributed Intelligence through Active Inference

While AIF as a concept has been developed and enhanced over recent years [Friston et al.(2024a), Friston et al.(2023a)], its transition towards applications is still at a relatively early stage [Friston et al.(2024b)]. Outside of neuroscience, AIF was mostly applied to robotics, as shown by the work of Lanillos et al. [Lanillos et al.(2021)] and Oliver at al. [Oliver et al.(2022)], who use it as a framework for sensing and perception. In particular, Oliver et al. give a comprehensive overview of how AIF allows (robotic) systems to act under uncertainty. Complementarily, De Vries et al. [de Vries(2023)] formulate a general design for AIF agents that applies across disciplines.

Nevertheless, applications of AIF to continuous stream processing systems are still scarce, with some exception, such as the works of Sedlak et al. [Sedlak et al.(2024a), Sedlak et al.(2024b)]. There, the authors show how to ensure SLOs of continuous video processing tasks by training a service interpretation model solely from observations; models were then shared between agents to speed up convergence to globally optimal solutions. In another work, Danilenka et al. [Danilenka et al.(2024)] developed an AIF agent that could optimize the SLO fulfillment of a federated learning task. Given the heterogeneity within the CC, they used AIF to find the optimal training configuration for individual clients. Levchuk et al. [Levchuk et al.(2019)] provide another example of a multi-agent system in which AIF is used to guide the exploration-exploitation tradeoff of multiple distributed agents. Noteworthy, they showed how agents' perception and collaboration could improve the convergence of sensing tasks.

Given the presented work, we conclude that AIF is regarded as a very promising solution for agent-based sensing environments, as can be found natively in the IoT domain. While some concepts of AIF have made their transition to machine learning and reinformcement learning [Mazzaglia et al.(2022), Tschantz et al.(2020a), Tschantz et al.(2020b)], wholesome solutions that encompass an entire software system are still scarce. To that extent, the research presented in this paper provides a well-needed guideline on how to design AIF agents in distributed computing scenarios. AIF can be of particular advantage in large-scale computing networks, like the CC, where empirically verifiable solutions are needed to prove the correctness of an approach.

8 Conclusions

In this article, we explored the use of AIF to distribute intelligence across a multi-service application within the CC. Specifically, we focused on a video stream processing pipeline requiring face blurring before delivery to the final consumer. This was achieved by deploying an AIF agent for each service and equipping them with mechanisms for elastic adaptation to service-specific requirements.

Our findings suggest that AIF is a promising approach for managing distributed applications in the CC. The AIF agents effectively fulfilled each service's SLOiDs by selecting targeted actions and, when necessary, adapting based on the outcomes of their decisions. When provided with expert-defined transition models, the agents achieved over 90% SLOiD fulfillment rates. In contrast, agents that learned their models through interaction with the environment achieved rates approximately 10% lower. When compared to state-of-the-art alternatives such as multi-agent reinforcement learning, we observed that extensively trained MARL agents slightly outperformed AIF agents. However, under minimal training conditions, AIF agents exhibited superior performance. We also analyzed agent behavior during service offloading scenarios, where the deployment environment changed. The AIF agents demonstrated adaptability by detecting the environmental shift and exploring new policies to recover fulfillment rates. Notably, agents learning their own transition models were capable of strategically deprioritizing unachievable SLOiDs in favor of maximizing achievable ones, showcasing a high degree of flexibility. Finally, we evaluated the computational cost of AIF agents in terms of execution time. A key factor in their efficiency is the selection of an appropriate policy length (i.e., look-ahead depth), as longer horizons can significantly increase computation time, especially in complex environments.

Looking ahead, AIF agents can be developed through a combination of expert knowledge, where the consequences of actions are predefined, and autonomous lifelong learning, where agents learn action outcomes over time. This hybrid design strategy supports the adaptability needed to address the heterogeneity and dynamism inherent in CC environments. We have also identified several gaps and promising directions for future research aimed at further aligning AIF capabilities with the unique demands of the CC. From a design perspective, further work is needed to enhance model flexibility, improve action predictability, and integrate more complex state-action relationships that better reflect real-world CC scenarios. On the runtime side, tackling challenges such as the action-state explosion and adapting models in response to environmental changes is critical for deploying robust and accountable AIF-driven services. Some of these challenges have been addressed in the broader AIF literature and could be adapted for CC applications. For instance, alternative planning algorithms have been proposed to enable faster [Champion et al.(2023), Paul et al.(2024)] and more sophisticated [Friston et al.(2021), Friston et al.(2023b)] action selection. Another promising direction is the incorporation of partially observed environments with a form of theory of mind, enabling agents to infer the internal states, intentions, and likely future actions of other agents [Albarracin et al.(2024)]. Combining these avenues will be crucial for modeling and managing the complex interactions that arise among large numbers of agents in distributed systems.

In summary, this study demonstrates that Active Inference offers a powerful foundation for distributed intelligence in the Computing Continuum. AIF agents can not only fulfill predefined service-level objectives but also adapt autonomously to evolving conditions. These findings lay the groundwork for future advancements in AIF-driven service orchestration and highlight the potential of AIF as a tool for enabling resilient, intelligent, and autonomous service management across the CC.

Acknowledgments

This work is supported by CNS2023-144359 financed by MICIU/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR.
Thanks to Dimitrije Markovic for his support.

References

[Akbari et al.(2024)] Negin Akbari, Adel N. Toosi, John Grundy, Hourieh Khalajzadeh, Mohammad S. Aslanpour, and Shashikant Ilager. 2024. iContinuum: An Emulation Toolkit for Intent-Based Computing Across the Edge-to-Cloud Continuum. IEEE Computer Society, 468–474. https://doi.org/10.1109/CLOUD62652.2024.00059

[Albarracin et al.(2024)] Mahault Albarracin, Riddhi J Pitliya, Toby St. Clere Smithe, Daniel Ari Friedman, Karl Friston, and Maxwell JD Ramstead. 2024. Shared Protentions in Multi-Agent Active Inference. *Entropy* 26, 4 (2024), 303.

[Baek et al.(2020)] Beomhan Baek, Joohyung Lee, Yuyang Peng, and Sangdon Park. 2020. Three Dynamic Pricing Schemes for Resource Allocation of Edge Computing for IoT Environment. *IEEE Internet of Things Journal* 7, 5 (May 2020), 4292–4303. https://doi.org/10.1109/JIOT.2020.2966627

[Beckman et al.(2020)] Pete Beckman, Jack Dongarra, Nicola Ferrier, Geoffrey Fox, Terry Moore, Dan Reed, and Micah Beck. 2020. Harnessing the computing continuum for programming our world. In *Fog Computing*, A. Zomaya, A. Abbas, and S. Khan (Eds.). John Wiley & Sons, Ltd, 215–230. https://onlinelibrary.wiley.com/doi/full/10.1002/9781119551713.ch7

[Beltran(2016)] Marta Beltran. 2016. Defining an Elasticity Metric for Cloud Computing Environments. In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS'15)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 172–179. https://doi.org/10.4108/eai.14-12-2015.2262685

[Beyer et al.(2016)] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems* (1st ed.). O'Reilly Media, Inc.

[Boutilier et al.(1999)] C. Boutilier, T. Dean, and S. Hanks. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11 (July 1999), 1–94. https://doi.org/10.1613/jair.575

[Casamayor Pujol et al.(2023)] Victor Casamayor Pujol, Andrea Morichetta, Ilir Murturi, Praveen Kumar Donta, and Schahram Dustdar. 2023. Fundamental Research Challenges for Distributed Computing Continuum Systems. *Information* 14, 3 (March 2023), 198. https://doi.org/10.3390/info14030198

[Casamayor Pujol et al.(2024)] Victor Casamayor Pujol, Boris Sedlak, Yanwei Xu, Praveen Kumar Donta, and Schahram Dustdar. 2024. DeepSLOs for the Computing Continuum. In *Proceedings of the 2024 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED'24)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3663338.3663681

[Champion et al.(2023)] Théophile Champion, Marek Grześ, Lisa Bonheme, and Howard Bowman. 2023. Deconstructing deep active inference. https://doi.org/10.48550/arXiv.2303.01618 arXiv:2303.01618 [cs].

[Collis et al.(2024)] Poppy Collis, Ryan Singh, Paul F. Kinghorn, and Christopher L. Buckley. 2024. Learning in Hybrid Active Inference Models. https://doi.org/10.48550/arXiv.2409.01066

[Danilenka et al.(2024)] Anastasiya Danilenka, Alireza Furutanpey, Victor Casamayor Pujol, Boris Sedlak, Anna Lackinger, Maria Ganzha, Marcin Paprzycki, and Schahram Dustdar. 2024. Adaptive Active Inference Agents for Heterogeneous and Lifelong Federated Learning. https://doi.org/10.48550/arXiv.2410.09099

[de Vries(2023)] Bert de Vries. 2023. Toward Design of Synthetic Active Inference Agents by Mere Mortals. arXiv:2307.14145 [nlin, stat].

[Dustdar et al.(2023)] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta. 2023. On Distributed Computing Continuum Systems. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (April 2023), 4092–4105. https://doi.org/10.1109/TKDE.2022.3142856

[Dustdar et al.(2011)] Schahram Dustdar, Yike Guo, Benjamin Satzger, and Hong-Linh Truong. 2011. Principles of Elastic Processes. *Internet Computing, IEEE* 15 (Nov. 2011), 66–71.

[Filinis et al.(2024)] Nikos Filinis, Ioannis Tzanettis, Dimitrios Spatharakis, Eleni Fotopoulou, Ioannis Dimolitsas, Anastasios Zafeiropoulos, Constantinos Vassilakis, and Symeon Papavassiliou. 2024. Intent-driven orchestration of serverless applications in the computing continuum. *Future Generation Computer Systems* 154 (May 2024), 72–86. https://doi.org/10.1016/j.future.2023.12.032

[Friston(2013)] Karl Friston. 2013. Life as we know it. *Journal of The Royal Society Interface* 10, 86 (Sept. 2013), 20130475. https://doi.org/10.1098/rsif.2013.0475

[Friston et al.(2021)] Karl Friston, Lancelot Da Costa, Danijar Hafner, Casper Hesp, and Thomas Parr. 2021. Sophisticated inference. *Neural Computation* 33, 3 (2021), 713–763.

[Friston et al.(2016)] Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, John O'Doherty, and Giovanni Pezzulo. 2016. Active inference and learning. *Neuroscience & Biobehavioral Reviews* 68 (Sept. 2016), 862–879. https://doi.org/10.1016/J.NEUBIOREV.2016.06.022

[Friston et al.(2024a)] Karl J Friston, Lancelot Da Costa, Alexander Tschantz, Alex Kiefer, Tommaso Salvatori, Victorita Neacsu, Magnus Koudahl, Conor Heins, Noor Sajid, Dimitrije Markovic, et al. 2024a. Supervised structure learning. *Biological Psychology* (2024), 108891.

[Friston et al.(2009)] Karl J. Friston, Jean Daunizeau, and Stefan J. Kiebel. 2009. Reinforcement Learning or Active Inference? *PLOS ONE* 4, 7 (July 2009), e6421.

[Friston et al.(2024b)] Karl J. Friston, Maxwell JD Ramstead, Alex B. Kiefer, Alexander Tschantz, Christopher L. Buckley, Mahault Albarracin, Riddhi J. Pitliya, Conor Heins, Brennan Klein, Beren Millidge, Dalton AR Sakthivadivel, Toby St Clere Smithe, Magnus Koudahl, Safae Essafi Tremblay, Capm Petersen, Kaiser Fung, Jason G. Fox, Steven Swanson, Dan Mapes, and Gabriel René. 2024b. Designing ecosystems of intelligence from first principles. *Collective Intelligence* (2024). https://doi.org/10.1177/26339137231222481

[Friston et al.(2023a)] Karl J Friston, Tommaso Salvatori, Takuya Isomura, Alexander Tschantz, Alex Kiefer, Tim Verbelen, Magnus Koudahl, Aswin Paul, Thomas Parr, Adeel Razi, et al. 2023a. Active inference and intentional behaviour. *arXiv preprint arXiv:2312.07547* (2023).

[Friston et al.(2023b)] Karl J Friston, Tommaso Salvatori, Takuya Isomura, Alexander Tschantz, Alex Kiefer, Tim Verbelen, Magnus Koudahl, Aswin Paul, Thomas Parr, Adeel Razi, et al. 2023b. Active inference and intentional behaviour. *arXiv preprint arXiv:2312.07547* (2023).

[Fürst et al.(2018)] Jonathan Fürst, Mauricio Fadel Argerich, Bin Cheng, and Apostolos Papageorgiou. 2018. Elastic Services for Edge Computing. In *2018 14th International Conference on Network and Service Management (CNSM)*. 358–362. https://ieeexplore.ieee.org/abstract/document/8584964?casa_token=39ojZ7iLIZEAAAAA:abK3nEI9LXbzy4GFAFGop2IytWTSZup_qrLa5It5LCaoh6dOfJvlu-qWzeJ2TXUlukKTWw ISSN: 2165-963X.

[Heins et al.(2022)] Conor Heins, Beren Millidge, Daphne Demekas, Brennan Klein, Karl Friston, Iain Couzin, and Alexander Tschantz. 2022. pymdp: A Python library for active inference in discrete state spaces. *Journal of Open Source Software* 7, 73 (May 2022), 4098. https://doi.org/10.21105/joss.04098

[Hu et al.(2023)] Shihong Hu, Weisong Shi, and Guanghui Li. 2023. CEC: A Containerized Edge Computing Framework for Dynamic Resource Provisioning. *IEEE Transactions on Mobile Computing* 22, 7 (July 2023), 3840–3854. https://doi.org/10.1109/TMC.2022.3147800

[Hu et al.(2020)] Xiaoyan Hu, Lifeng Wang, Kai-Kit Wong, Meixia Tao, Yangyang Zhang, and Zhongbin Zheng. 2020. Edge and Central Cloud Computing: A Perfect Pairing for High Energy Efficiency and Low-Latency. *IEEE Transactions on Wireless Communications* 19, 2 (Feb. 2020), 1070–1083. https://doi.org/10.1109/TWC.2019.2950632

[Kirchhoff et al.(2018)] Michael Kirchhoff, Thomas Parr, Ensor Palacios, Karl Friston, and Julian Kiverstein. 2018. The Markov blankets of life: autonomy, active inference and the free energy principle. *Journal of The Royal Society Interface* (2018).

[Lanillos et al.(2021)] Pablo Lanillos, Cristian Meo, Corrado Pezzato, Ajith Anil Meera, Mohamed Baioumy, Wataru Ohata, Alexander Tschantz, Beren Millidge, Martijn Wisse, Christopher L. Buckley, and Jun Tani. 2021. Active Inference in Robotics and Artificial Agents: Survey and Challenges. https://doi.org/10.48550/arXiv.2112.01871 arXiv:2112.01871 [cs].

[Levchuk et al.(2019)] Georgiy Levchuk, Krishna Pattipati, Daniel Serfaty, Adam Fouse, and Robert McCormack. 2019. Active Inference in Multiagent Systems: Context-Driven Collaboration and Decentralized Purpose-Driven Team Adaptation. In *Artificial Intelligence for the Internet of Everything*. Academic Press, Online.

[Liang et al.(2018)] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. https://arxiv.org/pdf/1712.09381

[Lin et al.(2018)] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E. Haque, Lingjia Tang, and Jason Mars. 2018. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 751–766. https://doi.org/10.1145/3173162.3173191

[Ma et al.(2024)] Huahong Ma, Bowen Ji, Honghai Wu, and Ling Xing. 2024. Video data offloading techniques in Mobile Edge Computing: A survey. *Physical Communication* 62 (Feb. 2024), 102261. https://doi.org/10.1016/j.phycom.2023.102261

[Makina et al.(2024)] Hela Makina, Asma Ben Letaifa, and Abderrezak Rachedi. 2024. Survey on security and privacy in Internet of Things-based eHealth applications: Challenges, architectures, and future directions. *SECURITY AND PRIVACY* 7, 2 (2024), e346. https://doi.org/10.1002/spy2.346

[Manaouil and Lebre(2020)] Karim Manaouil and Adrien Lebre. 2020. *Kubernetes and the Edge?* report. Inria Rennes - Bretagne Atlantique. https://inria.hal.science/hal-02972686

[Mariani and Zambonelli(2024)] Stefano Mariani and Franco Zambonelli. 2024. Distributed Discovery of Causal Networks in Pervasive Environments. In *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 1–6. https://doi.org/10.1109/PerComWorkshops59983.2024.10502971

[Mazzaglia et al.(2022)] Pietro Mazzaglia, Tim Verbelen, Ozan Catal, and Bart Dhoedt. 2022. The free energy principle for perception and action: A deep learning perspective. *Entropy* 24, 2 (2022), 301.

[Nardelli et al.(2024)] Matteo Nardelli, Gabriele Russo Russo, and Valeria Cardellini. 2024. Compute Continuum: What Lies Ahead?. In *Euro-Par 2023: Parallel Processing Workshops*. Springer Nature Switzerland, Cham, 5–17. https://doi.org/10.1007/978-3-031-50684-0_1

[Oliver et al.(2022)] Guillermo Oliver, Pablo Lanillos, and Gordon Cheng. 2022. An Empirical Study of Active Inference on a Humanoid Robot. *IEEE Transactions on Cognitive and Developmental Systems* 14, 2 (June 2022), 462–471. https://doi.org/10.1109/TCDS.2021.3049907

[Palacios et al.(2020)] Ensor Rafael Palacios, Adeel Razi, Thomas Parr, Michael Kirchhoff, and Karl Friston. 2020. On Markov blankets and hierarchical self-organisation. *Journal of Theoretical Biology* 486 (Feb. 2020).

[Parr and Friston(2018)] Thomas Parr and Karl J. Friston. 2018. The Discrete and Continuous Brain: From Decisions to Movement—And Back Again. *Neural Computation* 30, 9 (Sept. 2018), 2319–2347. https://doi.org/10.1162/neco_a_01102

[Parr et al.(2022)] Thomas Parr, Giovanni Pezzulo, and Karl J. Friston. 2022. *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. MIT Press. Google-Books-ID: UrZNEAAAQBAJ.

[Paul et al.(2024)] Aswin Paul, Noor Sajid, Lancelot Da Costa, and Adeel Razi. 2024. On efficient computation in active inference. *Expert Systems with Applications* 253 (2024), 124315.

[Pezzato et al.(2020)] Corrado Pezzato, Riccardo Ferrari, and Carlos Hernández Corbato. 2020. A Novel Adaptive Controller for Robot Manipulators Based on Active Inference. *IEEE Robotics and Automation Letters* 5, 2 (April 2020), 2973–2980. https://doi.org/10.1109/LRA.2020.2974451

[Schulman et al.(2017)] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. https://doi.org/10.48550/arXiv.1707.06347 arXiv:1707.06347 [cs].

[Sedlak et al.(2024a)] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. 2024a. Equilibrium in the Computing Continuum through Active Inference. *Future Generation Computer Systems* (May 2024). https://doi.org/10.1016/j.future.2024.05.056

[Sedlak et al.(2024b)] Boris Sedlak, Victor Casamayor Pujol, Andrea Morichetta, Praveen Kumar Donta, and Schahram Dustdar. 2024b. Adaptive Stream Processing on Edge Devices through Active Inference. https://doi.org/10.48550/arXiv.2409.17937 arXiv:2409.17937 [cs].

[Shahra et al.(2024)] Essa Q. Shahra, Wenyan Wu, Shadi Basurra, and Adel Aneiba. 2024. Intelligent Edge-Cloud Framework for Water Quality Monitoring in Water Distribution System. *Water* 16, 2 (Jan. 2024), 196. https://doi.org/10.3390/w16020196

[Shubha and Shen(2023)] Sudipta Saha Shubha and Haiying Shen. 2023. AdaInf: Data Drift Adaptive Scheduling for Accurate and SLO-guaranteed Multiple-Model Inference Serving at Edge Servers. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 473–485. https://doi.org/10.1145/3603269.3604830

[Spring et al.(2025)] Nikolaus Spring, Andrea Morichetta, Boris Sedlak, and Schahram Dustdar. 2025. MACH: Multi-Agent Coordination for RSU-centric Handovers. https://doi.org/10.48550/arXiv.2505.07827 arXiv:2505.07827 [cs].

[Sukhmani et al.(2019)] Sukhmani Sukhmani, Mohammad Sadeghi, Melike Erol-Kantarci, and Abdulmotaleb El Saddik. 2019. Edge Caching and Computing in 5G for Mobile AR/VR and Tactile Internet. *IEEE MultiMedia* 26, 1 (Jan. 2019), 21–30. https://doi.org/10.1109/MMUL.2018.2879591

[Tschantz et al.(2020a)] Alexander Tschantz, Manuel Baltieri, Anil K Seth, and Christopher L Buckley. 2020a. Scaling active inference. In *2020 international joint conference on neural networks (ijcnn)*. IEEE, 1–8.

[Tschantz et al.(2022)] Alexander Tschantz, Laura Barca, Domenico Maisto, Christopher L. Buckley, Anil K. Seth, and Giovanni Pezzulo. 2022. Simulating homeostatic, allostatic and goal-directed forms of interoceptive control using active inference. *Biological Psychology* 169 (March 2022), 108266. https://doi.org/10.1016/j.biopsycho.2022.108266

[Tschantz et al.(2020b)] Alexander Tschantz, Beren Millidge, Anil K Seth, and Christopher L Buckley. 2020b. Reinforcement learning through active inference. *arXiv preprint arXiv:2002.12636* (2020).

[Verma and Bala(2021)] Shveta Verma and Anju Bala. 2021. Auto-scaling techniques for IoT-based cloud applications: a review. *Cluster Computing* 24, 3 (Sept. 2021). https://doi.org/10.1007/s10586-021-03265-9

[Wu et al.(2023)] Yuxin Wu, Changjun Cai, Xuanming Bi, Junjuan Xia, Chongzhi Gao, Yajuan Tang, and Shiwei Lai. 2023. Intelligent resource allocation scheme for cloud-edge-end framework aided multi-source data stream. *EURASIP Journal on Advances in Signal Processing* 2023 (May 2023).

[Wu et al.(2021)] Zhanghao Wu, Eric Liang, Michael Luo, Sven Mika, Joseph E. Gonzalez, and Ion Stoica. 2021. RLlib Flow: Distributed Reinforcement Learning is a Dataflow Problem. In *Conference on Neural Information Processing Systems (NeurIPS)*. https://proceedings.neurips.cc/paper/2021/file/2bce32ed409f5ebcee2a7b417ad9beed-Paper.pdf

[Zafeiropoulos et al.(2024)] Anastasios Zafeiropoulos, Nikos Filinis, Eleni Fotopoulou, and Symeon Papavassiliou. 2024. AI-Assisted Synergetic Orchestration Mechanisms for Autoscaling in Computing Continuum Systems. *IEEE Communications Magazine* (2024). https://doi.org/10.1109/MCOM.001.2200583

[Zahidi et al.(2024)] Usman A. Zahidi, Arshad Khan, Tsvetan Zhivkov, Johann Dichtl, Dom Li, Soran Parsa, Marc Hanheide, Grzegorz Cielniak, Elizabeth I. Sklar, Simon Pearson, and Amir Ghalamzan-E. 2024. Optimising robotic operation speed with edge computing via 5G network: Insights from selective harvesting robots. *Journal of Field Robotics* n/a, n/a (July 2024). https://doi.org/10.1002/rob.22384

[Zhang et al.(2019)] Daniel (Yue) Zhang, Tahmid Rashid, Xukun Li, Nathan Vance, and Dong Wang. 2019. HeteroEdge: taming the heterogeneity of edge computing system in social sensing. In *Proceedings of the International Conference on Internet of Things Design and Implementation (IoTDI '19)*. Association for Computing Machinery, New York, NY, USA, 37–48. https://doi.org/10.1145/3302505.3310067

[Zhang et al.(2023)] Ziyang Zhang, Yang Zhao, and Jie Liu. 2023. Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster. In *Service-Oriented Computing*. Cham. https://doi.org/10.1007/978-3-031-48424-7_18

[Çatal et al.(2020)] Ozan Çatal, Samuel Wauthier, Cedric De Boom, Tim Verbelen, and Bart Dhoedt. 2020. Learning Generative State Space Models for Active Inference. *Frontiers in Computational Neuroscience* 14 (Nov. 2020). https://doi.org/10.3389/fncom.2020.574372

Appendix

A *Producer*'s transition model

WF Modality The transition probabilities for the **WF** modality specify the distribution $P(\mathbf{WF}_{t+1}|\mathbf{WF}_t, \mathit{FPS}_t, \mathit{Change\_FPS}_t)$. These probabilities depend on the current state of the modality ($\mathbf{WF}_t$), the current *FPS* state (FPS_t), and the relevant action component ($\mathit{Change\_FPS}_t$). Fully specifying the corresponding Conditional Probability Table (CPT) explicitly would require defining a 3x3 transition matrix for each of the 5 *FPS* values and 3 *Change_FPS* values (15 matrices total). However, we define the CPT completely and more compactly using the following rules:

1. If $\mathit{Change\_FPS}_t = \mathit{Stay}$, the state remains unchanged: $\mathbf{WF}_{t+1} = \mathbf{WF}_t$.
2. The state also remains unchanged if boundary conditions for *FPS* prevent the action's effect: i.e., if $\mathit{Change\_FPS}_t = \mathit{Increase}$ and FPS_t is already the highest value, or if $\mathit{Change\_FPS}_t = \mathit{Decrease}$ and FPS_t is already the lowest value.
3. If $\mathit{Change\_FPS}_t = \mathit{Increase}$ (and *FPS* is not maximum):
   - If $\mathbf{WF}_t = \mathit{Decrease}$ or $\mathbf{WF}_t = \mathit{Stay}$, then $\mathbf{WF}_{t+1} = \mathit{Decrease}$.
   - If $\mathbf{WF}_t = \mathit{Increase}$, then $\mathbf{WF}_{t+1} = \mathit{Stay}$.

4. If $\textit{Change\_FPS}_t = \textit{Decrease}$ (and *FPS* is not minimum):
   - If $\mathbf{WF}_t = \textit{Increase}$ or $\mathbf{WF}_t = \textit{Stay}$, then $\mathbf{WF}_{t+1} = \textit{Increase}$.
   - If $\mathbf{WF}_t = \textit{Decrease}$, then $\mathbf{WF}_{t+1} = \textit{Stay}$.

Note that these rules simplify the dependency on the *FPS* state, using it only to check boundary conditions. The precise dynamics might vary with specific hardware configurations in practice, but this expert model captures the intended behaviour.

CF & CR Modalities The transition dynamics for **CF** are defined analogously, depending on itself at t, $\textit{FPS}_t$, and $\textit{Change\_FPS}_t$. The dynamics for **CR** follow the same pattern but depend on $\mathbf{CR}_t$, $\textit{Resolution}_t$, and $\textit{Change\_Resolution}_t$.

FPS & Resolution Modailities The transitions for the *FPS* and *Resolution* modalities themselves are fully deterministic. These dynamics are defined by $P(\textit{FPS}_{t+1}|\textit{FPS}_t, \textit{Change\_FPS}_t)$ and $P(\textit{Resolution}_{t+1}|\textit{Resolution}_t, \textit{Change\_Resolution}_t)$, respectively. Their dependencies are shown in Figure 3, and their CPTs are defined by the following rules:

1. If the action (*Change_FPS* or *Change_Resolution*) is *Stay*, the value remains unchanged at $t+1$.
2. If the action is *Increase*, the value at $t+1$ transitions to the next higher discrete level defined for the modality, unless the value at t is already the maximum, in which case it remains unchanged.
3. If the action is *Decrease*, the value at $t+1$ transitions to the next lower discrete level, unless the value at t is already the minimum, in which case it remains unchanged.

B *Worker*'s transition model

Latency Modality The transition $P(\mathbf{Latency}_{t+1}|\mathbf{Latency}_t, \textit{Toggle\_Comm}_t)$ is simplified to depend only on the current state $\mathbf{Latency}_t$ and the action taken on the upstream communication channel ($\textit{Toggle\_Comm}_t$)[5]. The CPT is defined by the following deterministic rules:

1. If $\textit{Toggle\_Comm}_t = \textit{Enable}$, the state transitions to or remains *True*: $\mathbf{Latency}_{t+1} = \textit{True}$.
2. If $\textit{Toggle\_Comm}_t = \textit{Disable}$, the state remains unchanged: $\mathbf{Latency}_{t+1} = \mathbf{Latency}_t$.

This simplification assumes controlling the communication channel fully dictates this SLO status in our model. All state modalities affected by the action *Toggle_Comm* assume that the *Producer* agent will support the request sent by the other agents.

Execution Time Modality The transition $P(\textit{ExecTime}_{t+1}|\textit{ExecTime}_t, \textit{GPU}_t, \textit{Switch\_GPU}_t)$ depends on the current *Execution Time*, the current *GPU* state, and the action applied to the GPU. The CPT is defined by these rules:

1. If $\textit{Switch\_GPU}_t = \textit{Stay}$, or if $\textit{Switch\_GPU}_t = \textit{Switch off}$ when $\textit{GPU}_t = \textit{Off}$, or if $\textit{Switch\_GPU}_t = \textit{Switch on}$ when $\textit{GPU}_t = \textit{On}$. Then, the *Execution Time* remains unchanged ($\textit{ExecTime}_{t+1} = \textit{ExecTime}_t$).
2. If the GPU is on (i.e., $\textit{Switch\_GPU}_t = \textit{Switch on}$ and $\textit{GPU}_t = \textit{Off}$, leading to $\textit{GPU}_{t+1} = \textit{On}$), the *Execution Time* decreases by one discrete step ($\textit{ExecTime}_{t+1} = \text{prev}(\textit{ExecTime}_t)$). If $\textit{ExecTime}_t$ is already the lowest value, it remains unchanged.
3. If the GPU is off (i.e., $\textit{Switch\_GPU}_t = \textit{Switch off}$ and $\textit{GPU}_t = \textit{On}$, leading to $\textit{GPU}_{t+1} = \textit{Off}$), the *Execution Time* increases by one discrete step ($\textit{ExecTime}_{t+1} = \text{next}(\textit{ExecTime}_t)$). If $\textit{ExecTime}_t$ is already the highest value, it remains unchanged.

FPS Modality The transition $P(\textit{FPS}_{t+1}|\textit{FPS}_t, \textit{Toggle\_Comm}_t)$ depends on the current FPS value and the action on the upstream communication channel.

1. If communication is disabled or unchanged ($\textit{Toggle\_Comm}_t = (\textit{Disable}$ or $\textit{Stay})$), the FPS remains the same: $\textit{FPS}_{t+1} = \textit{FPS}_t$.
2. If communication is enabled ($\textit{Toggle\_Comm}_t = \textit{Enable}$), the *FPS* decreases by one discrete step: $\textit{FPS}_{t+1} = \text{prev}(\textit{FPS}_t)$. If $\textit{FPS}_t$ is already the lowest value, it remains unchanged.

[5] **Latency** also depends on the *Execution Time* and the *FPS*, however, here we focus on the communication action to simplify all definitions.

W-consumption Modality The $P(\textbf{W-consumption}_{t+1}|\textbf{W-consumption}_t, \textit{ShareInfo}_t, \textit{GPU}_t, \textit{Toggle\_Comm}_t, \textit{Switch\_GPU}_t)$ depends on the current **W-consumption** state, the states of *Share Information* and *GPU*, and the actions applied to them. We assume the resulting **W-consumption** state depends on whether the *ShareInfo* and *GPU* states are *effectively activated or deactivated* by the actions taken. The CPT rules are:

1. If both *Share Information* and *GPU* are activated (transition to or stay *True*), **W-consumption** increases by one step: $\textbf{W-consumption}_{t+1} = \text{next}(\textbf{W-consumption}_t)$.
2. If one modality (*ShareInfo* or *GPU*) is activated while the other's state remains unchanged, **W-consumption** increases by one step: $\textbf{W-consumption}_{t+1} = \text{next}(\textbf{W-consumption}_t)$.
3. If either modality is deactivated (transitions to or stays *False*), **W-consumption** decreases by one step: $\textbf{W-consumption}_{t+1} = \text{prev}(\textbf{W-consumption}_t)$.
4. If one modality is activated and the other is deactivated within the same time step, **W-consumption** remains the same: $\textbf{W-consumption}_{t+1} = \textbf{W-consumption}_t$.
5. If both modalities' states remain unchanged, **W-consumption** remains the same: $\textbf{W-consumption}_{t+1} = \textbf{W-consumption}_t$.

Share Info Modality The transition $P(\textit{ShareInfo}_{t+1}|\textit{ShareInfo}_t, \textit{Toggle\_Comm}_t)$ depends on the current state and the action on the upstream communication channel. The CPT rules are:

1. If the action is *Disable*, the state becomes False: $\textit{ShareInfo}_{t+1} = \textit{False}$.
2. If the action is *Enable*, the state becomes True: $\textit{ShareInfo}_{t+1} = \textit{True}$.
3. If the action is *Stay*, the state remains unchanged: $\textit{ShareInfo}_{t+1} = \textit{ShareInfo}_t$.

GPU Modality The transition $P(\textit{GPU}_{t+1}|\textit{GPU}_t, \textit{Switch\_GPU}_t)$ depends on the current state and the action applied to the GPU. The CPT rules mirror those for *Share Info*, using actions *Switch on*, *Switch off*, and *Stay*:

1. If $\textit{Switch\_GPU}_t = \textit{Switch off}$, then $\textit{GPU}_{t+1} = \textit{False}$.
2. If $\textit{Switch\_GPU}_t = \textit{Switch on}$, then $\textit{GPU}_{t+1} = \textit{True}$.
3. If $\textit{Switch\_GPU}_t = \textit{Stay}$, then $\textit{GPU}_{t+1} = \textit{GPU}_t$.

C *Consumer*'s transition model

Success Modality The transition $P(\textbf{Success}_{t+1}|\textbf{Success}_t, \textit{Resolution}_t, \textit{Toggle\_Comm}_t)$ depends on the current **Success** state, the *Resolution*, and the upstream communication action. The CPT is defined by the following rules, which currently simplify the dependency on $\textit{Resolution}_t$:

1. If $\textit{Toggle\_Comm}_t$ is *Enable*, the state transitions to or remains *True*: $\textbf{Success}_{t+1} = \textit{True}$.
2. If $\textit{Toggle\_Comm}_t$ is *Disable* or *Stay*, the state remains unchanged: $\textbf{Success}_{t+1} = \textbf{Success}_t$.

Smoothness Modality The transition $P(\textbf{Smoothness}_{t+1}|\textbf{Smoothness}_t, \textit{FPS}_t, \textit{Toggle\_Comm}_t)$ depends on the current **Smoothness** state, the *FPS* state, and the upstream communication action. The CPT rules are:

1. If $\textit{Toggle\_Comm}_t$ is *Disable* or *Stay*, the state remains unchanged: $\textbf{Smoothness}_{t+1} = \textbf{Smoothness}_t$.
2. If $\textit{Toggle\_Comm}_t$ is *Enable*: the resulting state $\textbf{Smoothness}_{t+1}$ becomes *True*, reflecting the expectation that enabling communication aims to improve this SLO, potentially by facilitating an FPS increase.

C-consumption Modality The transition $P(\textbf{C-consumption}_{t+1}|\textbf{C-consumption}_t, \textit{ShareInfo}_t, \textit{Toggle\_Comm}_t)$ depends on the current **C-consumption** state, the current *Share Information* state, and the upstream communication action. The CPT rules are:

1. If $\textit{Toggle\_Comm}_t = \textit{Enable}$ causes *Share Information* to switch from *False* to *True* (e.g., $\textit{ShareInfo}_t = \textit{False}$), then **C-consumption** increases: $\textbf{C-consumption}_{t+1} = \text{next}(\textbf{C-consumption}_t)$.
2. If the action $\textit{Toggle\_Comm}_t = \textit{Disable}$ causes *Share Information* to switch from *True* to *False* (e.g., $\textit{ShareInfo}_t = \textit{True}$), then **C-consumption** decreases: $\textbf{C-consumption}_{t+1} = \text{prev}(\textbf{C-consumption}_t)$.
3. In all other cases, **C-consumption** remains unchanged: $\textbf{C-consumption}_{t+1} = \textbf{C-consumption}_t$.

FPS and Resolution Modalities The transitions $P(\mathit{FPS}_{t+1}|\mathit{FPS}_t, \mathit{Toggle\_Comm}_t)$ and $P(\mathit{Resolution}_{t+1}|\mathit{Resolution}_t, \mathit{Toggle\_Comm}_t)$ depend on their respective current states and the upstream communication action. The deterministic CPT rules are:

1. If $\mathit{Toggle\_Comm}_t$ is *Disable* or *Stay*, the respective modality's value remains unchanged ($\mathit{FPS}_{t+1} = \mathit{FPS}_t$, $\mathit{Resolution}_{t+1} = \mathit{Resolution}_t$).
2. If $\mathit{Toggle\_Comm}_t$ is *Enable*, both the *FPS* and *Resolution* values increase by one discrete step, if possible (i.e., $\mathit{FPS}_{t+1} = \text{next}(\mathit{FPS}_t)$, $\mathit{Resolution}_{t+1} = \text{next}(\mathit{Resolution}_t)$), subject to saturation at their maximum values.

Share Information Modality The transition probabilities $P(\mathit{ShareInfo}_{t+1}|\mathit{ShareInfo}_t, \mathit{Toggle\_Comm}_t)$ are defined identically to the corresponding modality in the *Worker* agent model.