# Adaptive Stream Processing on Edge Devices through Active Inference

Boris Sedlak[1*], Victor Casamayor Pujol[2], Andrea Morichetta[1], Praveen Kumar Donta[3], Schahram Dustdar[1,2]

[1]Distributed Systems Group, TU Wien, Vienna, 1040, Austria.
[2]Distributed Intelligence and Systems-Engineering Lab (DISL), Universitat Pompeu Fabra, Barcelona, 08018, Spain.
[3]Department of Computer and Systems Sciences, Stockholm University, Stockholm, 16425, Sweden.

*Corresponding author. E-mail: b.sedlak@dsg.tuwien.ac.at;
Contributing authors: v.casamayor@dsg.tuwien.ac.at; a.morichetta@dsg.tuwien.ac.at; praveen@dsv.su.se; dustdar@dsg.tuwien.ac.at;

**Abstract**

The current scenario of IoT is witnessing a constant increase on the volume of data, which is generated in constant stream, calling for novel architectural and logical solutions for processing it. Moving the data handling towards the edge of the computing spectrum guarantees better distribution of load and, in principle, lower latency and better privacy. However, managing such a structure is complex, especially when requirements, also referred to Service Level Objectives (SLOs), specified by applications' owners and infrastructure managers need to be ensured. Despite the rich number of proposals of Machine Learning (ML) based management solutions, researchers and practitioners yet struggle to guarantee long-term prediction and control, and accurate troubleshooting. Therefore, we present a novel ML paradigm based on Active Inference (AIF) – a concept from neuroscience that describes how the brain constantly predicts and evaluates sensory information to decrease long-term surprise. We implement it and evaluate it in a heterogeneous real stream processing use case, where an AIF-based agent continuously optimizes the fulfillment of three SLOs for three autonomous driving services running on multiple devices. The agent used causal knowledge to gradually develop an understanding of how its actions are related to requirements fulfillment, and which configurations to favor. Through this approach, our agent requires up to thirty iterations to converge to the optimal solution,

showing the capability of offering accurate results in a short amount of time. Furthermore, thanks to AIF and its causal structures, our method guarantees full transparency on the decision making, making the interpretation of the results and the troubleshooting effortless.

**Keywords:** Active Inference, Machine Learning, Edge Intelligence, Service Level Objectives, Markov Blanket

# 1 Introduction

Recent years have reported a constant transition of logic and computation from the central cloud towards the edge of the network [1], bringing the execution closer to the Internet of Things (IoT) devices that actually generate data. The reasons behind migrating the computation to the edge span from saving bandwidth, to improving privacy, as well as speeding up data processing (i.e., decreasing latency). Furthermore, this transition, facilitated by improvements in hardware and connectivity, currently allows the training and deployment of Machine Learning (ML), up to Deep Learning (DL) models. This step is pivotal in such scenarios, as ML models play an essential role in interpreting and predicting the behavior of applications and distributed systems, in order to guarantee Quality of Service (QoS). Indeed, in the last years ML has been vastly utilized [2] for the management of complex, distributed infrastructures and applications, with tasks that go from estimating the impact of redeployment [3] to forecasting potential system failures [4].

However, contrarily to the cloud infrastructure, deploying services on top of resource-restricted edge devices while guaranteeing their QoS is not trivial. Indeed, edge devices, which span from commodity servers to dedicated machines, have a constrained pool of resources. Typically, a set of common strategies involve letting edge devices scale their services through local reconfiguration, assisted by ML [5]. Despite the validity of these strategies, most of them do not consider when the initial conditions changed. This is especially limiting for ML models, which are not retrained although new observations would be available [3, 4], inevitably leads to a drift in observing and predicting the system state. Exemplifying it, imagine an elastic computing system, as envisioned in [6], which observes the system through a set of metrics, evaluates whether QoS requirements – also called Service Level Objectives (SLOs) – are fulfilled, and dynamically reconfigures the system to ensure SLOs are met. If the variable distribution changes and the ML model is not adjusted, this makes it impossible to interpret system metrics correctly, and any consequential reconfiguration will fail to fulfill its purpose. Therefore, a solution to guarantee the precision of ML models over time involves implementing continuous feedback mechanisms; this could, for example, be achieved by optimizing a value function, as in reinforcement learning (RL) [7, 8]. However, approaches like RL are slow to converge and typically computationally intensive. Furthermore, leveraging ML algorithms means, nowadays, often relying on DL models. Despite their great performance in various, complex applications, most of them suffer from a lack of interpretability, which is instead essential in

core tasks like distributed infrastructure and application management. For these reasons, we believe that this scenario requires a more holistic approach, which starts with making the SLOs first-class citizens during ML training. Further, any component that uses ML for inference should actively resolve or report ambiguities. We envision that such a level of self-determination could be provided by Active Inference (AIF), a concept from neuroscience that describes how the brain constantly predicts and evaluates sensory information to decrease long-term surprise. In cases where ML training and inference are carried out in close proximity to the data source, i.e., on edge devices, AIF can ensure model accuracy whenever the accuracy drops. Equipped with AIF, edge devices could continuously infer system configurations that ensure QoS. Furthermore, AIF allows to develop causal understanding of a process; this raises the trust for inferred results [3, 9].

In this paper, we extend our previous contribution [10], where we presented a design study of an AIF agent optimizing the throughput in a smart factory. To show the general applicability of our approach, we now apply AIF extensively for distributed processing systems and provide a novel evaluation with 3 Edge-based stream services showing how the agents are able to adapt service configuration to match QoS requirements. Agents operate autonomously and decentralized while ensuring the SLO compliance on their local edge devices. At its core, the agent follows an action-perception cycle where it first estimates which parameter assignments would violate given SLOs, then compares this expectation with new observations, and finally, adjusts its beliefs (i.e., the ML model) accordingly. While exploring the value space, it favors solutions that are likely to improve the model precision; this, in turn, provides the agent with a clear understanding of the causal relations between model variables. Hence, the contributions of this article are:

- An adaptive stream processing mechanism based on Active Inference that continuously optimizes the QoS of streaming pipelines through local reconfiguration; thus, processing services scale autonomously according to environmental impacts
- A transparent decision-making process for AIF agents that adjusts processing configurations according to expected SLO fulfillment and model improvement. The underlying causal structures make results empirically verifiable and increase trust.
- The evaluation of the presented AIF agent under three different stream processing services and heterogeneous edge device types. This underlines how the presented methodology is apt to support a wider range of stream processing scenarios.

The remainder of the paper is structured as follows: Section 2 provides background information on AIF principles in edge computing; Section 3 presents related work; in Section 4 we outline the design process of an AIF agent, which we implement and evaluate in Section 5. Finally, Section 6 concludes the paper.

## 2 Background

To dynamically adjust stream processing, the core mechanism applied in this paper is AIF; given that AIF is a concept that originates from neuroscience, we use this section to summarize core concepts of AIF according to Friston et al. [11–14]. This includes

(1) a high-level picture of how AIF works, (2) an illustrative example of how agents use AIF, and (3) a formal representation of how AIF agents adjust their beliefs and choose actions that fulfill their preferences. Following that, we delineate our view of the intersection between AIF and distributed computing systems, highlighting how AIF concepts are relevant for stream processing on edge devices.

## 2.1 Free Energy Principle and Active Inference

The Free Energy Principle (FEP) and its corollary Active Inference (AIF) stem from neuroscience, aiming at answering the overarching question: *how does the brain work?*

**The high-level picture.** Seeking to answer such a fundamental and complex question, Friston offers a theory [15] of an overarching structure for the brain and cognition. In their understanding, cognitive agents have the capacity to build an *internal model* of their observed environment and the underlying *generative processes*. This internal model – also called a *generative model* – is used by agents to *predict and adjust* their environment according to their *preferences*. In the FEP theory, and especially in AIF, the system revolves around two core elements: the *generative process* and the *generative model*. The first is the underlying causal structure of the environment's behavior that produces agent's observations; the *generative model* approximates, from the agent's perspective, the behavior of the environment.

Using their generative model, agents predict and modify their environment; for this, they assume that the generative model aligns closely (i.e., in terms of KL divergence) with the generative process. However, if the generative process and the model differ, this discrepancy will "surprise" the agent, causing it to adjust its model closer towards the process [16]. This surprise (after Bayesian surprise [17]) plays a fundamental part in Friston's theoretical framework because *Free Energy (FE)* formally presents an upper bound on surprise. To minimize FE, and hence surprise, AIF agents constantly engage in action-perception cycles, where they (1) predict sensory inputs, observe the environment, and update their beliefs depending on the outcome. Afterward, they (2) actively adjust the world to their preferences. Internally, agents organize their generative models in hierarchical structures; each level interprets lower-level causes and, based on that, provides predictions to higher levels. This process of using existing beliefs (widely known as priors) to calculate the probability of related events is commonly known as Bayesian inference [18] and it allows agents to improve their understanding of the environment. AIF models are structured by self-contained Bayesian structures, known as Markov Blankets (MBs), that separate the internal states of a system from its outside environment, formalizing the action-perception cycle.

**An illustrative example.** To exemplify this approach, imagine an individual who believes that it is raining; as a consequence, he will take an umbrella to avoid getting wet, as this would otherwise be uncomfortable. As defined before, individual agents interpret observable processes through generative models. In this case, an individual might reason that it is raining as he observes water drops falling from the sky. Using its generative model, the individual infers with high probability that it is raining; the respective action is to take an umbrella to fulfill its preference of staying dry. However, imagine that the observed water drops were actually caused by a neighbor

watering their plants. Thus, the generative model and the process diverge, and the agent is "surprised" once he leaves the house. More generally, individuals can either take pragmatic action, e.g., picking an umbrella to fulfill their preferences, or otherwise, improve decision-making by exploring the environment through epistemic actions. For example, looking at the blue sky and the neighbor's balcony reveals that drops resulted from watering the plants, avoiding the later consequence of carrying an umbrella on a sunny day. The agent thus updates its prior beliefs (i.e., rain → water) according to new information (i.e., rain → water ← flowers) to form its posterior beliefs.

**Formalization.** Picking up from the explanations of AIF and FE, it remains to provide a formal representation of these concepts: given that an agent equipped with generative model $m$ makes an observation $o$, the surprise $\Im(o|m)$, as shown in Eq. (1), is the negative log-likelihood of the observation [13]. The FE of the model – expressed as the Kullback-Leibler divergence ($KL$) between approximate posterior probability ($Q$) of hidden states ($x$) and their exact posterior probability ($P$) – is an upper-bound on surprise. This is formalized in Eq. (2); however, the exact posterior probability ($P$) is practically intractable, hence, $Q$ is used as an approximation. The FEP now uses variational inference [19] as the method to find the best approximation that minimizes the difference between both distributions.

$$\Im(o|m) = -\ln \overbrace{P(o|m)}^{\text{Model Evidence}} \tag{1}$$

$$F[Q,o] = \underbrace{KL[Q(x)||P(x|o,m)] + \Im(o|m)}_{\text{(Variational) Free Energy}} \geq \Im(o|m) \tag{2}$$

AIF agents use this mathematical framework to select among actions [13, 20]; for this, they condition all terms on the policy ($\pi$). However, instead of looking into the approximate model performance, AIF agents select the best policy according to possible future states, actions, and observations. More formally, agents minimize their Expected Free Energy (EFE) by selecting the optimal policy. The EFE can be expressed as in Eq (3):

$$EFE = -\overbrace{E_{Q(x|\pi)}[\ln P(x|R)]}^{\text{pragmatic value}} - \overbrace{E_{Q(x,z|\pi)}[\ln Q(z|x,\pi) - \ln Q(z|\pi)]}^{\text{information gain}} \tag{3}$$

where the first term, i.e., the pragmatic value ($pv$), expresses how likely an action will produce observations that fulfill an agent's preferences (R). The $pv$ pushes an agent to take actions that will satisfy its goal; for a computing system, this means fulfilling its SLOs. Noteworthy, preferred observations can be understood as a reward function in reinforcement learning [8, 21, 22]. The second term, i.e., the information gain ($ig$), pushes an agent towards actions that allow it to learn about its environment; in the literature, $ig$ is also called "epistemic value". This expression shows that minimizing EFE intrinsically deals with the classic exploration-exploitation trade-off.

## 2.2 AIF Principles in Distributed Systems

Large-scale distributed computing systems, especially when handling streams of data and events coming from a wide range of devices and applications, consist of many interconnected components; therefore, modeling and managing [23, 24] such structures requires ingenuity. In this regard, we identify three central challenges of distributed systems and stream processing, which can benefit from AIF concepts. First, (1) how to model large-scale systems and underlying structures to understand their overall functionality, e.g., to find the root cause of a bottleneck in the data processing pipeline. Second, (2) how to mitigate SLO violations within distributed systems under unexpected runtime dynamics; this step is essential when we manage real-time, latency-sensitive scenario like in stream processing. All in all, the main goal boils down to taking the best actions at a given point in time. Finally, (3) how to maintain long-term fulfillment of the system's requirements through effective management; this step implies taking care of variations in the infrastructure and in the behavior and quality of data. While there exist several ML-based methods for distributed computing management [25], most of these approaches leave several of these challenges still open. In the following, we align these challenges with the main characteristics of AIF, elucidating how AIF can fundamentally improve the management of distributed computing systems.

### 2.2.1 System Modeling through Causal Structures

Understanding the behavior of distributed computing systems and reasoning on the effects of individual actions is challenging due to the entanglement and complexity among individual components. For example, finding the root cause of abnormal behaviors or backtracing which configuration decision in which part of the system propagated a certain system state is not trivial and is often computationally intractable. In this scenario, using causal models in distributed systems [26] is a captivating solution that can help identify the source of specific behaviors.

In this direction, a tool commonly used is Bayesian Networks [27] (BN). It can be trained to identify connections and dependencies between components through conditional probability functions. While BNs offer a good understanding of correlation within a system, as Pearl points out [18, 28], only observational data (i.e., data that we collect from a system without knowing what is its current state) is not enough to gain causal knowledge. Here, AIF plays an important role: when AIF agents start intervening in their environment, the agents gain more valuable insights into the system behavior. Repeating this action-perception cycle can enrich BN, gradually adding information that turns them into actual causal graphs.

In contrast to solely relying on Deep Learning methods, introducing causal structures to management solutions has the fundamental advantage of guaranteeing an interpretation of system behavior and the effects of management actions or recommendations, thus improving trustworthiness [29]. Specifically, in distributed computing systems, a causal structure can explain how metrics (e.g., latency or CPU load) are related to the system state [9], backtrack which service or device caused a system failure [3], or predict the impact of redeployment [30].

### 2.2.2 Mitigating SLO Violations through Free Energy Minimization

Modeling SLO fulfillment through BN or causal graphs, as explained above, is challenging. Replicating the probability distribution of events in large-scale systems is, at the very least, complex. Hence, performing statistical inference is often intractable due to the distribution's complexity. Therefore, as shown in Eq (2), AIF leverages variational inference (VI) [19]: VI uses the evidence lower bound (ELBO) to make the problem an optimization one. Without entering into details, increasing ELBO minimizes KL divergence and, therefore, allows to approximate the probability distribution. This problem can obviously be solved together with gradient methods and mechanisms like Deep Neural Networks. Indeed, ELBO and its variants are very popular in ML and DL (a notable example being Variational Auto Encoders [31]).

Minimizing KL-Divergence is crucial in AIF; this means, that agents intervening in the environment can lead to a better understanding of the environment and, as a consequence, makes it more likely to achieve agents' preferences. Naturally, illustrating this process inevitably draws a parallel with Reinforcement Learning (RL); however, AIF presents fundamental differences [22]: whereas in RL, agents maximize a reward function, in AIF the agent intervenes in the environment to minimize EFE. This difference, albeit subtle, offers essential advantages. Minimizing EFE means not only performing the action that leads to observing the desired output (pragmatic value) but also improving the precision of the generative model (information gain).

In distributed computing systems, epistemic actions often suffice to reduce uncertainties about expected outcomes: distributed systems can resolve contextual information by identifying a low-utilized agent for tasks offloading [32, 33], or evaluating resource availability before scaling a system [5, 34]. In other cases, taking action aids to prevent or mitigate SLO violations, e.g., in case of resource allocation or task offloading [35–38]. This embraces the common tradeoff between seeking either pragmatic value (exploitation) or epistemic value (exploration). Multi-agent systems [39] control this through hyperparameters, which foster early exploration of a value space but decay over time as agents report little improvement.

### 2.2.3 Long-term System Balance through Homeostasis

The ultimate goal for an AIF agent is to constantly minimize the surprise and thus persist over time; this requires guaranteeing that certain internal variables remain within defined ranges. In cybernetic theories of living organisms, this process is called *homeostasis*. For example, the human body requires a core temperature of approximately 37° to ensure internal chemical processes. Hence, modeling the environment accurately allows agents to make the right decisions that fulfill its requirements, e.g., picking an umbrella when it's raining so that the body does not cool down.

The human body, as a complex system, has distinct mechanisms to ensure internal requirements, e.g., in case of cooling down, it can raise the core temperature through shivering. The fact that shivering happens unconsciously underlines how systems, including distributed processing systems, benefit from autonomous requirements assurance. While processing systems can also pose requirements in terms of (CPU) temperature, these and other requirements are specified as Service Level Objectives

(SLOs) that must be assured during processing. Common instances of SLOs are QoS requirements, such as response time or availability [6]; in case SLOs are violated, processing systems can resolve this through elasticity strategies [40, 41], e.g., cloud computing scales computational resources to limit response time. Complex systems can have multiple elasticity strategies at their disposal [34], e.g., computing systems generally scale three elasticity dimensions: resources, quality, and cost. To optimize SLO fulfillment through local reconfiguration, existing works [9, 42] used the notation of Markov blankets, which encompass relations between system variables and elasticity strategies. Thus, the agent can infer how to best ensure its requirements. Simply put, an agent's preferred observation is having SLOs fulfilled; in case SLOs are violated, the agent will take action to correct this by choosing an adequate strategy according to the Markov blanket model.

# 3 Related Work

In this section, we delineate the state of the art in adaptive stream processing and its relation to AIF. First, we depict the current scenario in AIF, where, despite the praiseworthy efforts to theorize the framework, only a few contributions focus on implementing it, and still not in interdisciplinary scenarios. Later, we offer a snapshot of the prevailing approaches for adaptive stream processing and how approaches based on AIF can improve the QoS during stream processing.

***AIF Applications***

While, to the best of our knowledge, there exists no off-the-shelf implementation of AIF in distributed systems; a handful of research works have combined AIF with computer science:

The authors in [43] discuss AIF as a general computational framework, highlighting how existing research used AIF for (simulating) sensory processing. Touching on the design of AIF agents, Heins et al. [44] provide a Python simulation that exemplifies how to structure action-perception cycles. Heins et al. further remark that existing AIF research largely focuses on formally constructing models in isolated environments [20] such as Matlab SPM rather than putting them into action, e.g., to improve the precision of ML models. Thus, a more hands-on application of AIF is to extend reinforcement learning with AIF principles [7, 8]. However, most research to date either uses only a few AIF principles or is not applied enough to easily transfer presented concepts to distributed systems.

The work in [39] is, therefore, an exception because it embeds AIF into the IoT and describes how AIF can improve the behavior of adaptive agents. Thus, individual agents may dynamically regroup into hierarchical teams, federate knowledge, and collectively strive after a common goal (i.e., a search task). By emphasizing the information exchange between agents, they were able to speed up the convergence of the distributed task. However, while they focused on FE minimization, they did not treat the other two principles we identified for AIF in distributed systems: causal inference and homeostasis. In this paper, we will present an agent that uses all three AIF principles to infer actions, maintain agents' internal equilibrium, and persist over time.

***Runtime adaptation of stream services***

Runtime adaptation of stream services is an extensive topic in the scientific literature. We narrow down the related work by leveraging the taxonomy of adaptation mechanisms presented by Cardellini et al. [45]. In that regard, we will focus on works that perform processing adaptations. The initial driver for that boosted research related with the adaptation of stream services stems from commercial large-scale video streaming services that emerged more than a decade ago. For instance, Huang et al. [46] showed how to achieved 10-20% less rebuffering rate while achieving a higher video rate. Conversely to this research, our work focuses on computing paradigms outside the Cloud, aiming at adapting services that are at the Edge, next to IoT devices.

Leaning on more recent research, Khani et al. [47] presents a similar use case to ours, i.e., real-time video inference on Edge devices, but adapts the machine learning models deployed at the Edge devices by distilling knowledge from a centralized model. Further, they use a server, which contains the centralized model, to also adapt the frame sampling rate for each device. In contrast, our work aims toward a completely decentralized architecture, providing the Edge service the full autonomy to adapt.

Ma et al. [48] develop a system to adapt video bitrate for live video streaming at the Edge based on Quality-of-Experience (QoE) requirements. They use deep reinforcement learning (DRL) to adapt the bitrate and distribute the video traffic to the clients according to their QoE needs. Similarly, Cao et al. [49] build a DRL-based system to adjust video streams for Edge computing according to the expected QoE. Interestingly, they develop a programming model to facilitate the potential adaptations of the streams. Further, as discussed by Cao et al. [49], DRL models require periodical re-training as the data shifts while AIF includes this re-training within its normal formulation thanks to the possibility of performing epistemic adaptations.

Similarly, as described by Furst et al. [5], our work consists of achieving elastic services for stream processing at the Edge. Further we use SLOs to autonomously adapt each service. In contrast to Furst et al., our approach uses AIF to adapt stream processing services, which is a more flexible perspective for dynamic and long-lasting operations because AIF improves its behavior over time.

***Takeaways***

Despite existing contributions in the field of stream adaptation, we argue that our approach can offer better control in a more generalizable way. First, conversely, to previous stream services, we focus on defining ways to map SLOs for the autonomous control of a distributed computing system. Secondly, we manage that using AIF, which provides the flexibility for managing services that dynamically change over time. Finally, with our work, we contribute to the AIF field by both providing an implementation of AIF strategies and showing its potential in a real use case.

# 4 AIF Adaptive Streaming Agent

In this Section, we delineate the three main steps for assembling an adaptive AIF agent for stream processing: (1) setting up tools and methods for observing the behavior of a continuous stream; (2) based on these outputs, training a generative model to predict

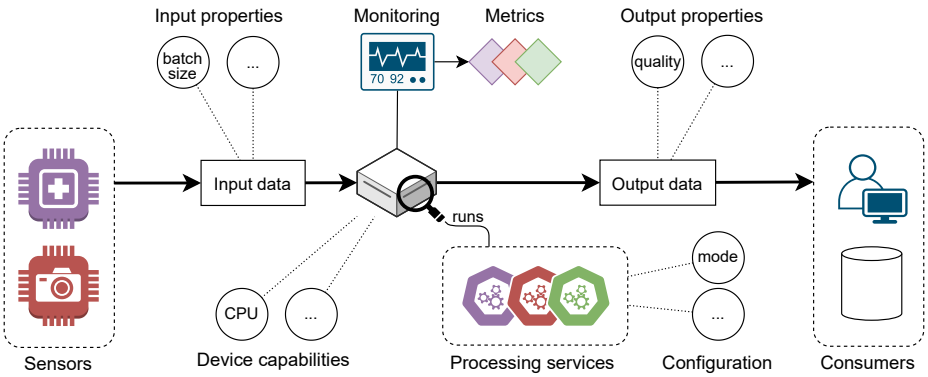



**Fig. 1**: Abstract representation of a continuous stream processing scenario; sensors provide input data that is processed by services located at an edge device; processing is observable through metrics and respective results are provided to stream consumers

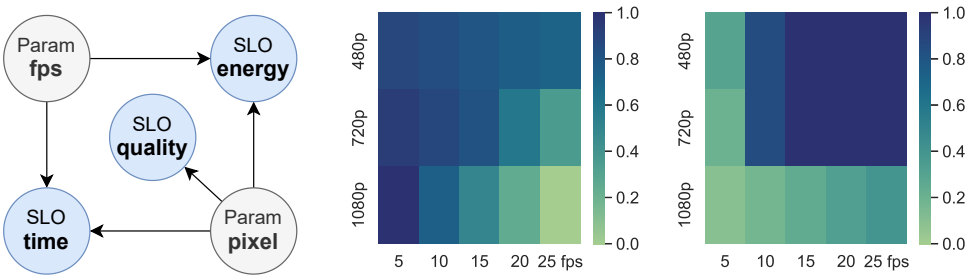
and interpret stream processing behaviors; (3) implement the agent's algorithm for executing the AIF cycle, i.e., the application of the generative model for both action and perception. These three steps form a coherent, generalizable pipeline, applicable to different streaming use cases.

## 4.1 Continuous Stream Processing

Modeling a streaming service requires a well-defined architecture; first, there is the need of a transparent view of its internal processes, including the characteristics of the in- and out-flow data, e.g., batch size or data type in input, and quality of the processed output. Additionally, it is essential to have visibility on quantifiable metrics about the fulfillment of requirements. For example, the capabilities, e.g., embedded CPU or GPU, of the edge devices that process the data have a strong impact on the performance. Similarly, the strategy for the service configuration ($c$) is relevant. Service configuration allows the node manager to set, or limit, the local computational resources, achieving elasticity in a constrained setting, but also impacting the SLOs' fulfillment; e.g., using a low-processing mode benefits energy efficiency, but limits the performance. Therefore, we directly monitor the performance on the device, providing a continuous stream of metrics ($D$) for an immutable list of variables. Figure 1 summarizes this architecture, with its main components; in this work we focus on exploring the case of IoT sensors data stream.

## 4.2 Generative Model Construction

Interpreting stream processing boils down to one central activity: reasoning about metrics. For this, we apply generative models, as introduced for AIF. Our implementation of the generative process follows the design by Parr et al. [13], adapting it for the stream processing use case. Following [13], we focus on the three main generative model design aspects:




(a) Variable relations (b) Pragmatic value (c) Information gain

**Fig. 2**: Conditional (or causal) variable relations encoded in a Bayesian network; variable states (i.e., *fps* and *pixel*) form a 2D solution space where each parameter combination features a distinct pragmatic value (*pv*) and information gain (*ig*)

**A**. What are the generative model's *components*, and what its *interfaces*?
**B**. What is the *hierarchical* and *temporal depth* of the generative model?
**C**. What *probabilities* are *encoded* in the model, and how are they *updated* over time?

**A.** We envision a generative model consisting of two combined components: a directed graph that expresses variable relationships, and the precise conditional probabilities of which variable states are likely observed under certain environmental conditions. We represent the underlying graph as a Directed Acyclic Graph (DAG), where the edges indicate conditional dependency. For instance, consider Fig. 2a, where the edge *fps* $\rightarrow$ *energy* shows how the number of frames per second (fps), i.e., a configuration parameter, has a decisive impact on the energy consumption of the processing device. If we specify an SLO for minimizing *energy consumption*, such as **energy** $\leq 15W$, it is possible to infer the probability of fulfilling it under different service configurations, i.e., according to the precise assignments of *fps* and *pixel*.

While applying the generative model appears straightforward, training it requires high-quality observations; one method to build it is through Bayesian Network Learning (BNL), e.g., as applied by [3, 50, 51]. In the absence of training data, it is also possible to specify a BN through expert knowledge [52]. Given the resulting generative model, agents can reason how likely it is to observe SLO violations under a certain service configuration, e.g., exceeding **energy** with $fps = 15$. However, if the generative model does not reflect the generative process accurately, the agent can be surprised by the actual outcome. As defined by Dustdar et al. [53], improving the model's accuracy requires a clear understanding of the *interfaces* between the agent and the environment as these will map the observations from the generative process to internal elements of the generative model.

**B.** The temporal and hierarchical depth are two more sophisticated properties of the generative model. For the given use case, the temporal depth is bounded by the length of the policy $\pi$, i.e., how many steps in the future the behavior is predicted. Given that longer policies have a higher complexity for predicting the outcome, the

policy length can be chosen in accordance with the evaluation frequency. This means, that for stream processing, it can be desirable to evaluate the SLO fulfillment with high frequency (e.g., every 500ms) and conversely keep the policy length low.

The hierarchical depth, on the other hand, is bounded by the number of variables and states in the generative model; for instance, Figures 2b and 2c show a parameter space that consists of two variables, i.e., *fps* and *pixel*, that can take 5 and respectively 3 variable states. While this solution space and the DAG in Figure 2a appears minimalistic, BNs can grow up to thousands of variables [54]; depending on the granularity, it is possible to split up processes into increasingly smaller substructures, which can each be constrained and managed by a more specific set of SLOs [55]. Hence, maintaining a desired level of abstraction [12] ensures lower model complexity for inference and training. In that regard, extracting the MB around SLO variables [42] helps reduce the number of variables that must be considered.

**C.** When an agent's observation presents a discrepancy with the expected outcome, e.g., an SLO is violated despite the previous agent intervention to have a proper configuration, it generate a "surprise," calling the agent to adjust its generative model. Here, the priors are both the variable relations and the conditional probabilities encoded in the generative model. This means, that upon observing something that goes contrary to its expectation, the agent updates its model to form its posterior beliefs. While variable relations and conditional probabilities are subject to retraining, we assume the list of monitored variables to be immutable. Nevertheless, given that feature-evolving streams are gaining popularity [56], extending this to mutable lists of variables promises a valuable asset.

For the generative model presented in this paper, we distinguish two types of variables: (1) the configuration parameters (i.e., parents in Figure 2a), which decide *how* streaming data is processed, and (2) the SLOs (i.e., child nodes), which constrain *whether* the stream processing behavior fulfills the agent's preferences. The classification into parameters stems from the system definition, hence, it is immutable and determined depending on which variables the agent can adjust locally. However, for remaining variables, the decision to turn them into SLOs, can also be taken at a later stage. Thus, it is possible to adjust the list of SLOs and, more importantly, also their desired thresholds. For instance, depending on the type of processing device, we can decide to cap **energy** either by 10W or 15W. Thereby, we described the characteristics and boundaries of the generative model as required to interpret continuous stream processing.

## 4.3 Active Inference Cycle

To continuously ensure model accuracy during stream processing, and consequently high SLO fulfillment, we embed the generative model into the action-perception loop executed by an AIF agent. Figure 3 gives a high-level overview of this action-perception loop: a stream processing service (red) is executed on an edge device; during this execution, the monitored metrics are provided to the AIF agent. Application stakeholders (e.g., DevOps engineers) can specify SLOs that should be ensured during processing; these SLOs constitute the agent's preferences of what it would like to observe. As the AIF agent receives a batch of metrics (i.e., observations), it evaluates the degree to

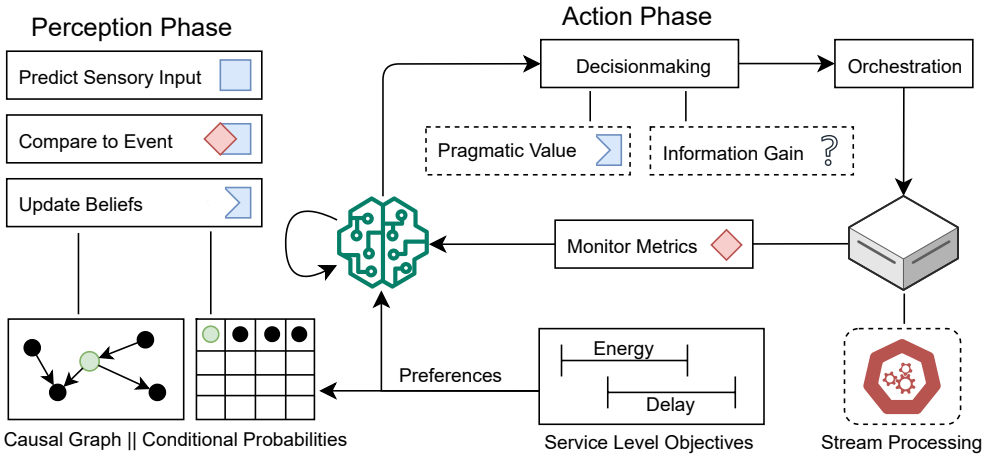



**Fig. 3**: High-level action-perception cycle in AIF; stream processing metrics are interpreted using a generative model; the agent updates the model according to prediction errors and makes adjustments by balancing pragmatic value with information gain

which SLOs were fulfilled according to Eq. (4):

$$\phi(Q) = \frac{\sum_{i=1}^{|Q|} \phi(q_i)}{|Q|} \tag{4a}$$

$$\phi(q_i) = \phi(q_i, d \mid \forall d \in D_{q_i}) = \sum_{j=1}^{|D_{q_i}|} \frac{\phi(d_j, q_i)}{|D|} \tag{4b}$$

$$\text{where } \phi(q_i, d_j) = \begin{cases} 1, & \text{if } d_{j_{\min}}^{q_i} \leq d_j \leq d_{j_{\max}}^{q_i} \\ 0, & \text{otherwise} \end{cases} \tag{4c}$$

where $Q$ represents a set of SLOs, $D$ the list metrics, and $\phi$ the resulting SLO fulfillment. As shown in Eq. (4c), the overall SLO fulfillment is determined by the ratio of samples that are found in the desired range, delimited by the thresholds. The actual SLO fulfillment is then compared to the agent's prediction according to its generative model; when the two diverge, the agent adjusts the generative model, which means retraining the causal graph and the variables' conditional probabilities.

To minimize EFE through optimal action, agents consider the known concepts of either (1) changing the environment toward its preferences to maximize pragmatic value ($pv$) or (2) improving decision-making by resolving contextual information to maximize information gain ($ig$). In the following, we introduce how we calculate a

simplified version of the information gain as shown below in Eq. (5):

$$\Im(D \mid m) = \sum_{i=1}^{|D|} -\log P(d_i \mid m) \tag{5a}$$

$$ig(c) = \left( \frac{\tilde{\Im}_c}{\tilde{\Im}_\omega} \right) \times 100 \tag{5b}$$

where the *ig* of a configuration $c$ is calculated based on the surprise that this configuration caused in the past. If a configuration repeatedly shows surprising results, this indicates that the agent cannot yet predict its *pv* accurately; consequently, this assigns a high *ig* to it. While Eq. (1) provided the idea of how to calculate the surprise ($\Im$) for a single observation, Eq. (5a) shows how the surprise for a batch of metrics equals the sum of the individual metrics. Hence, when calculating the *ig* for a configuration, e.g., $fps = 15$ and $pixel = 480$ in Figure 2c, the agent considers how the median surprise ($\tilde{\Im}_c$) for the configuration relates to the global median surprise ($\tilde{\Im}_\omega$). The weights that the agent assigns to *pv* and *ig* are subject to hyperparameter optimization; to speed up convergence, we assign *pv* double the weight of *ig*, i.e., $w_{pv} = 2 \times w_{ig}$.

Up to this point, explanations of *pv* and *ig* were built on the assumption that the generative model can always provide an approximate probability for any possible state. However, the generative model might not be able to provide this approximation without having observations for a certain parameter configuration. Further, recall that we are dealing with discrete states, thus, even though an agent infers based on observations how likely it is to fulfill SLOs with $fps = \{10, 20\}$, this lacks understanding of what will happen if $fps = 15$. Hence, we interpolate between the *pv* and *ig* of the closest neighbors, or for the first initial values, choose the closest neighboring value. Notice, how the interpolation took place in Figures 2b and 2c; this becomes particularly important because the agent in its simplest form minimizes EFE (i.e., *pv* and *ig*) by comparing a potentially exponential number of parameter combinations. While the combinatorial complexity requires dedicated optimization, the interpolation between configurations provides a workaround in cases when no observations are available.

As soon as the agent has come to a decision, it has to orchestrate the action, i.e., by adjusting the local processing configuration. Thus, the action-perception cycle is closed; whenever the agent receives a new batch of observations, it repeats this cycle by evaluating the actual SLO fulfillment, adjusting its generative model according to the new observations, and using its generative model to infer a service adaptation.

# 5 Evaluation

In the following section, we present the implementation and evaluation of our methodology over a heterogenous set of actual services and real edge devices. To underline the generality of our approach, we showcase how our stream processing framework fits multiple use cases; the overarching goal is to evaluate whether AIF agents can dynamically find a satisfying stream processing configuration. For each use case, we document the experimental setup, including the service implementations and applied

processing hardware. Finally, we present the experimental results and conclude with a critical discussion.

## 5.1 Implementation

To embed the designed AIF agent into actual processing devices, we provide a Python-based prototype that comprises both the generative model construction as well as the continuous action-perception cycle. The prototype of the AIF agent, the implementation of the processing services, and the experimental results are all shared in one publicly accessible repository[1]. Thus, we aim to make our results reproducible and allow others to interact with our methodology.

To isolate resource consumption, we execute each processing service in a Python thread. During that time, each service observes its own SLO fulfillment as part of its action-perception cycle; in the present state, this is done every 2000ms, though it can be customized for different service types. To avoid interfering with the stream processing task, the training of the generative model is detached from the processing thread. To train and update its generative model, AIF agents use pgmpy [57], a Python library for Bayesian Network Learning (BNL). As shown in previous work [9, 42], pgmpy is very effective for providing both the structure and the conditional dependencies of BNs, making it apt for the continuous action-perception cycle.

## 5.2 Experimental Setup

To evaluate our prototype we provide three use cases for stream processing, showing for each of them how our framework can ensure its processing requirements. Table 1 provides essential information on the three processing services, such as the number of configuration parameters and SLOs. In the following, we describe each of them in more detail, including their practical relevance and application:

**CV.** Videos rank among the most commonly streamed data types; hence, Computer Vision (CV) gained increasing popularity for detecting objects in videos [58] or ensuring privacy [59] by transforming the video content. Hence, the first processing service is a video inference task based on Yolov8 [60] that detects vehicles, traffic lights, and pedestrians in a traffic junction.

**LI.** Sticking to road traffic and transportation, autonomous driving is empowered by multitudes of sensors embedded in Autonomous Vehicles (AVs). As such, Lidar is commonly used to scan AVs' surroundings through point cloud processing [61]; AVs use this to react in real-time to dynamic traffic conditions. Hence, the second service processes point clouds through Lidar (LI) using the SFA3D [62] library.

**QR.** Within recent years, QR codes have quickly found their way into our everyday lives, e.g., to share contact information or hyperlinks. However, QR codes are also applied to track objects [63], i.e., by attaching the code to an object and continuously inferring its position. Hence, the third service uses OpenCV [64] to track the position of a vehicle according to a QR code attached to its rear.

[1]The prototype of the stream processing framework is available at GitHub, accessed on July 31st 2024

**Table 1**: List of all stream processing services covered by the framework

| ID | Service Description | CUDA | Parameters | SLOs |
|---|---|---|---|---|
| *CV* | Object Detection with Yolov8 [60] | Yes | *pixel*, *fps* | **time**, **energy**, **rate** |
| *LI* | LiDAR Point Cloud Processing [62] | Yes | *mode*, *fps* | **time**, **energy** |
| *QR* | Detect QR Code w/ OpenCV [64] | No | *pixel*, *fps* | **time**, **energy** |

(a) CV (Yolov8) (b) QR (OpenCV) (c) LI (SFA3D)

**Fig. 4**: Demo output for each service according to prerecorded input data; all three services are iteratively processing video streams (CV and QR) or binary input (LI)

Figure 4 shows a demo output for each service to make more tangible what contents are processed and produced by the three services. Notice, that to ensure a stable evaluation environment, the service processed either prerecorded videos (*CV* & *QR*) or binary-encoded point clouds (*LI*). For adjusting the outcome, each service has specific configuration parameters available, such as the resolution (*pixel*) and *fps* for *CV* and *QR*; *LI* accepts an additional parameter *mode* to define the point cloud radius. While *pixel* and *fps* are common properties of video streams, we assume that the AIF agent can still change these during processing, i.e., by shrinking the maximum video resolution or sampling the video to a lower frame rate.

We empirically map each service's expected QoS level to a list of SLOs, based on our expert knowledge. After several experiments, which are out of the scope of this work, the following values proved especially useful: we constrain the processing **time** to $\leq 1000/fps$ (i.e., frames must be processed faster than they come in). To ensure efficient stream processing, the maximum **energy** consumption is capped at $\leq 15W$. For what concerns the video resolution (*pixel*) provided to *CV*, the service uses the respective Yolov8 model size (i.e., v8n, v8s, v8m). However, this affects the number of objects that are detected, which is ensured through the **rate** SLO. The presented configuration parameters and SLOs describe the list of variables that will be included in the generative model; nevertheless, only through BNL it is possible to extract their relations and dependencies.

We evaluate on two different instances of Nvidia Jetson boards, namely Jetson Orin *NX* and Orin *AGX*, which are described in more detail in Tab. 2. Since these devices have different processing capabilities, local AIF agents cannot infer parameter configurations using one global model because the mismatch between generative model and device will likely cause the agent to be surprised. Hence, this device heterogeneity requires AIF agents to train their generative models separately. Further, to decrease

**Table 2**: List of edge devices involved in the evaluation; depending on the Jetson power mode, devices have different numbers of CPU and GPU cores available

| ID | Full Device Name | Mode | Price[2] | CPU | RAM | GPU | CUDA |
|---|---|---|---|---|---|---|---|
| $AGX_+$ | Jetson Orin AGX | MAX | 800 € | ARM 12C | 64 GB | Volta 8k | 12.2 |
| $AGX_-$ | Jetson Orin AGX | LIM | 800 € | ARM 8C | 64 GB | Volta 4k | 12.2 |
| $NX_+$ | Jetson Orin NX | MAX | 450 € | ARM 8C | 8 GB | Volta 4k | 11.4 |
| $NX_-$ | Jetson Orin NX | LIM | 450 € | ARM 4C | 8 GB | Volta 2k | 11.4 |

**Table 3**: Preferred configuration per service x device combination

| | *CV* | *QR* | *LI* |
|---|---|---|---|
| $AGX_+$ | $\langle 1080\text{p}, 5\text{ fps}\rangle = 0.94$ | $\langle 720\text{p}, 15\text{ fps}\rangle = 1.0$ | $\langle \text{single}, 5\text{ fps}\rangle = 0.98$ |
| $AGX_-$ | $\langle 720\text{p}, 15\text{ fps}\rangle = 0.62$ | $\langle 720\text{p}, 5\text{ fps}\rangle = 1.0$ | $\langle \text{single}, 5\text{ fps}\rangle = 0.93$ |
| $NX_+$ | $\langle 720\text{p}, 10\text{ fps}\rangle = 0.83$ | $\langle 720\text{p}, 5\text{ fps}\rangle = 1.0$ | $\langle \text{single}, 5\text{ fps}\rangle = 0.92$ |
| $NX_-$ | $\langle 480\text{p}, 5\text{ fps}\rangle = 0.73$ | $\langle 480\text{p}, 10\text{ fps}\rangle = 1.0$ | $\langle \text{single}, 5\text{ fps}\rangle = 0.90$ |

energy consumption, Jetson boards offer the feature of operating in different device modes, which limits the available device resources. However, this introduces further heterogeneity, even between identical devices operating in different modes; hence, the list of devices contains for both devices two entries: $AGX_+$ indicates that the device operated without constraints, and $AGX_-$ indicates that the local resources were limited. Notice, how switching between device modes reduces the number of CPU and GPU cores for both device types.

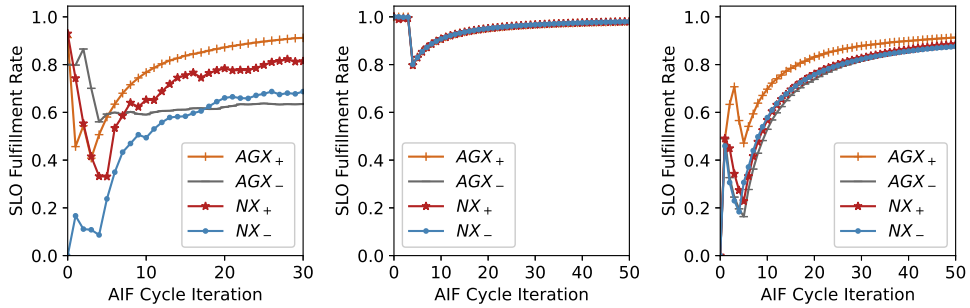
While the Jetsons' specific version of Nvidia CUDA has only minor importance, CUDA itself is crucial to speed up video or point cloud processing through GPU acceleration. However, to the best of our knowledge, there are no off-the-shelf solutions to use CUDA for scanning QR codes with OpenCV; hence, CUDA was only used to accelerate *CV* and *LI*, while *QR* was processed entirely on the CPU.

## 5.3 Results & Discussion

Given the experimental setup, we evaluate the prototype based on: (1) how long does the AIF agent to find a satisfying parameter configuration; (2) is the behavior of the AIF agent explainable during its action-perception cycle; (3) how does the service and device heterogeneity impact agents' decisions. For this, we conduct a total number of 12 experiments, which stems from the fact that we evaluate each services on each device type. During the experiments, we capture the current SLO fulfillment, the preferred configuration, and the *pv* and *ig* values that the AIF agent assigns to each parameter combination. Additionally, we collect the structure of the generative model.

Figure 5 shows the SLO fulfillment that the AIF agents reported during each of its iterations. For each of the services, the tendency is that the agent initially reports low SLO fulfillment while it is still exploring the solution space; however, after 20-30 iterations the agent converges to a clear preference. Table 3 shows the respective

[2]Prices adopted from sparkfun, accessed Jul 31st 2024




(a) CV (Yolov8) (b) QR (OpenCV) (c) LI (SFA3D)

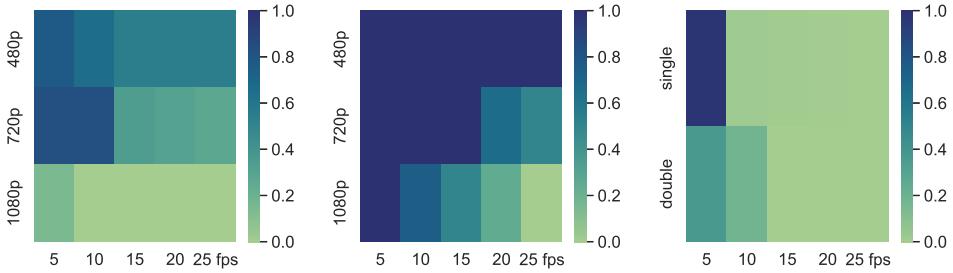
**Fig. 5**: Empirical SLO fulfillment measured during service execution; initial training rounds show unstable behavior, while later rounds converge to a clear preference

parameter configurations to which the agents converged. For 11 out of 12 cases, we could verify through exhaustive comparison that the chosen parameter configuration was optimal, or showed only minor divergence. For the $CV$ service on $AGX_{-}$ (grey line in Figure 5a), however, the exploration did not provide the desired results, and so it converged to a sub-optimal configuration.

Given this, we conclude that the AIF agent is not guaranteed to find the optimal solution; however, the results showed that in most of the cases it was still possible to find the optimal solution. Hence, using the AIF agent to supervise the processing service showed promising results. Combined with its low processing overhead [42], this makes it a great fit for for resource-constrained edge devices.

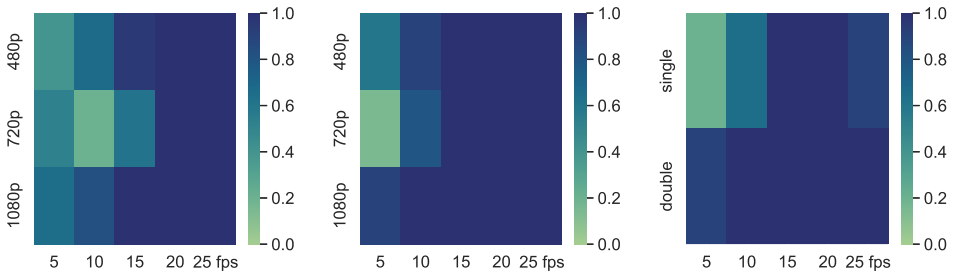
To analyze whether the behavior of the AIF agent is explainable, we retrace its steps and verify how it selects its preferred parameter combinations. In Figures 6 and 7, we provide the final matrices for the $pv$ and $ig$ values that the agent assigned to each parameter configuration at the end of the experiment. Recall, that in case a configuration was not empirically evaluated yet, the agent interpolates its $pv$ and $ig$ according to neighboring configurations. Given the figures, we notice that the matrices are coherent with the preferred configurations in Table 3: the chosen configuration showed both a high $pv$ due to the expected SLO fulfillment, but also a low $ig$ due to the increasing exploitation, i.e., the configuration promise little model improvement. Additionally, Figure 8 shows the structure of the generative models for each of the processing services, that is, how the AIF agent linked the configuration parameters and SLO variables during BNL. It is according to this variable relations, that the agent reasons how it must adjust *pixel* and *fps* to fulfill the SLOs.

Given that, we conclude that the agents' behavior is rationally explainable because the edges in the DAG are logical and match our expert knowledge. Using this graph, it is possible to explain why the agent converged to a certain configuration, e.g., $CV$ at $NX_{+}$ to $pixel = 5$ and $fps = 5$, because this configuration showed clearly the highest $pv$ value in Figure 6a. Still, although its $ig$ value in Figure 7a was lower, the agent decided that no further exploration was needed. Notice how in both Figures 6a and 6b, there are multiple parameter configurations that promise equal $pv$ values; however,




(a) CV (Yolov8) (b) QR (OpenCV) (c) LI (SFA3D)

**Fig. 6**: PV Matrices for the three services executed at $NX_+$; individual cells in the heatmap show the expected SLO fulfillment for each combination of config parameters

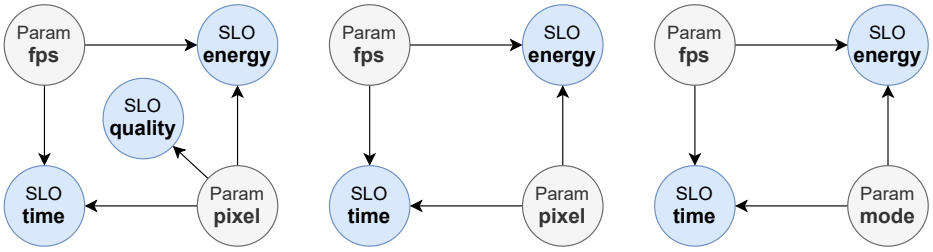



(a) CV (Yolov8) (b) QR (OpenCV) (c) LI (SFA3D)

**Fig. 7**: IG Matrices for the three services executed at $NX_+$; individual cells in the heatmap indicate expected model improvement when using the specific configuration

the agent randomly selects one of them because they all fulfill its preferences equally. Given that we would specify an SLO that minimizes *energy*, not just cap it, this would lead to a more specific preference here. Together, these rules form coherent patterns that can be empirically verified, thus increasing the trust in agents' results.

To answer the third research question, we use the existing results. Given those, we conclude that the presented framework for adaptive stream processing showed promising results for all combinations of heterogeneous devices and services. In particular, Table 3 underlines how the agents adjusted their local generative model to find configurations that match their device capabilities, e.g., $AGX_+$ chose $pixel = 1080p$ for $CV$ and hence optimized the **rate** SLO, whereas $NX_-$'s capabilities only sufficed to choose $pixel = 480p$. Given all that, we see strong potential for using the agent to optimize the QoS of other stream processing use cases; the requirements for this are known from Section 4: observable stream processing and clear variable specifications, i.e., parameters and SLOs. However, to make this framework apt for more complex scenarios, further evaluations are needed that operate with larger solution spaces.




(a) CV (Yolov8) (b) QR (OpenCV) (c) LI (SFA3D)

**Fig. 8**: Directed Acyclic Graphs (DAGs) reflecting the variable relations in the generative models; relations were extracted through BNL and are empirically verifiable

# 6 Conclusion

This paper presented a novel framework for adaptive stream processing that continuously ensures QoS during processing on resource-constrained edge devices. Processing services were supervised through Active Inference, a behavioral framework from neuroscience, which allows agents to develop and maintain a logical model of how to ensure processing requirements, also called SLOs. Thus, edge devices can ensure that stream processing complies with requirements. By executing AIF agents directly on edge devices, it is possible to observe with low latency how different service configurations impact SLO fulfillment. Depending on the outcome, AIF agents continuously adjust their generative model and use it to infer service configurations that promise high SLO fulfillment or further model improvement. To that extent, the AIF agents make use of causal variable relations that determine how their actions affect SLO fulfillment; this increases the trustworthiness of inferred configurations.

To evaluate the framework, we implemented a Python-based AIF agent for monitoring and guaranteeing SLO fulfillment, by adapting local processing configurations. In this way, we implemented an elasticity strategy on constrained edge devices, that are otherwise unable to scale resources as in cloud computing scenarios. We evaluated our framework on three different scenarios for stream processing, two for video processing and one for Lidar sensors, which were executed on heterogeneous edge devices. Not only did the devices, i.e., two instances of Nvidia Jetson, have different processing capabilities, but they could also operate in different hardware modes. Given this setup, the problem was to find service configurations that fulfill SLOs on processing time, energy consumption, and detection rate. Our results showed that the AIF agents required roughly 20 to 30 iterations in the action-perception cycle to converge to the optimal solution; further, the inferred results were empirically verifiable and followed logical patterns. Based on that, we see strong potential for AIF agents to support elastic stream processing in further, real time scenarios. We plan to extend the evaluation of our contribution to more complex scenarios, where the solution space is larger. Furthermore, we plan to have an in depth comparison of how AIF performs when compared to other ML methods.

## Funding

This work has been supported by the European Union's Horizon Europe research and innovation program under grant agreements No. 101135576 (INTEND) and No. 101070186 (TEADAL).

## Data availability

The datasets used in this article are cited appropriately.

## Conflict of interest

Authors declare that there are no conflicts of interest.

## References

[1] Deng, S., Zhao, H., Fang, W., Yin, J., Dustdar, S., Zomaya, A.Y.: Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. IEEE Internet of Things Journal (2020) https://doi.org/10.1109/JIOT.2020.2984887

[2] Hua, H., Li, Y., Wang, T., Dong, N., Li, W., Cao, J.: Edge computing with artificial intelligence: A machine learning perspective. ACM Computing Surveys **55**(9), 1–35 (2023)

[3] Chen, P., Qi, Y., Hou, D.: CauseInfer: Automated End-to-End Performance Diagnosis with Hierarchical Causality Graph in Cloud Environment. IEEE Transactions on Services Computing (2019)

[4] Morichetta, A., Pujol, V.C., Nastic, S., Pusztai, T., Raith, P., Dustdar, S., Vij, D., Xiong, Y., Zhang, Z.: Demystifying deep learning in predictive monitoring for cloud-native SLOs (2023)

[5] Fürst, J., Fadel Argerich, M., Cheng, B., Papageorgiou, A.: Elastic Services for Edge Computing. In: 2018 14th International Conference on Network and Service Management (CNSM), pp. 358–362 (2018)

[6] Nastic, S., Morichetta, A., Pusztai, T., Dustdar, S., Ding, X., Vij, D., Xiong, Y.: SLOC: Service Level Objectives for Next Generation Cloud Computing. IEEE Internet Computing **24**(3) (2020)

[7] Martínez, E.C., Kim, J.W., Barz, T., Cruz, M.: Probabilistic Modeling for Optimization of Bioreactors using Reinforcement Learning with Active Inference. Computer Aided Chemical Engineering (2021)

[8] Friston, K.J., Daunizeau, J., Kiebel, S.J.: Reinforcement Learning or Active Inference? PLOS ONE **4**(7), 6421 (2009)

[9] Sedlak, B., Pujol, V.C., Donta, P.K., Dustdar, S.: Designing Reconfigurable Intelligent Systems with Markov Blankets. In: Service-Oriented Computing, pp. 42–50 (2023). https://doi.org/10.1007/978-3-031-48421-6_4

[10] Sedlak, B., Pujol, V.C., Donta, P.K., Dustdar, S.: Active Inference on the Edge: A Design Study. In: 2024 IEEE PerCom Workshops, pp. 550–555 (2024). https://doi.org/10.1109/PerComWorkshops59983.2024.10502828

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

[12] Kirchhoff, M., Parr, T., Palacios, E., Friston, K., Kiverstein, J.: The Markov blankets of life: autonomy, active inference and the free energy principle. Journal of The Royal Society Interface (2018)

[13] Parr, T., Pezzulo, G., Friston, K.J.: Active Inference: The Free Energy Principle in Mind, Brain, and Behavior. The MIT Press, ??? (2022)

[14] Friston, K.J., Ramstead, M.J., Kiefer, A.B., Tschantz, A., Buckley, C.L., Albarracin, M., Pitliya, R.J., Heins, C., Klein, B., Millidge, B., Sakthivadivel, D.A., Smithe, T.S.C., Koudahl, M., Tremblay, S.E., Petersen, C., Fung, K., Fox, J.G., Swanson, S., Mapes, D., René, G.: Designing ecosystems of intelligence from first principles. Collective Intelligence (2024) https://doi.org/10.1177/26339137231222481

[15] Friston, K.: The free-energy principle: a unified brain theory? Nature reviews neuroscience **11**(2), 127–138 (2010)

[16] Bruineberg, J., Rietveld, E., Parr, T., Maanen, L., Friston, K.J.: Free-energy minimization in joint agent-environment systems: A niche construction perspective. Journal of theoretical biology **455**, 161–178 (2018)

[17] Itti, L., Baldi, P.: Bayesian surprise attracts human attention. Vision research **49**(10), 1295–1306 (2009)

[18] Pearl, J.: Causal inference in statistics: An overview. Statistics Surveys **3**(none), 96–146 (2009) https://doi.org/10.1214/09-SS057

[19] Blei, D.M., Kucukelbir, A., McAuliffe, J.D.: Variational Inference: A Review for Statisticians. Journal of the American Statistical Association (2017) https://doi.org/10.1080/01621459.2017.1285773

[20] Smith, R., Friston, K.J., Whyte, C.J.: A step-by-step tutorial on active inference and its application to empirical data. Journal of Mathematical Psychology **107**, 102632 (2022) https://doi.org/10.1016/j.jmp.2021.102632

[21] Tschantz, A., Millidge, B., Seth, A.K., Buckley, C.L.: Reinforcement Learning

through Active Inference. arXiv (2020)

[22] Sajid, N., Ball, P.J., Parr, T., Friston, K.J.: Active inference: demystified and compared. Neural Computation **33**(3), 674–712 (2021)

[23] Firmani, D., Leotta, F., Mathew, J.G., Rossi, J., Balzotti, L., Song, H., Roman, D., Dautov, R., Husom, E.J., Sen, S., *et al.*: Intend: Intent-based data operation in the computing continuum. In: CEUR WORKSHOP PROCEEDINGS, vol. 3692, pp. 43–50 (2024). CEUR-WS

[24] Morichetta, A., Lackinger, A., Dustdar, S.: Cohabitation of intelligence and systems: Towards self-reference in digital anatomies. In: 2024 IEEE International Conference on Service-Oriented System Engineering (SOSE) (2024). IEEE

[25] Ilager, S., Muralidhar, R., Buyya, R.: Artificial intelligence (ai)-centric management of resources in modern distributed computing systems. In: 2020 IEEE Cloud Summit, pp. 1–10 (2020). IEEE

[26] Pujol, V.C., Sedlak, B., Donta, P.K., Dustdar, S.: On Causality in Distributed Continuum Systems. IEEE Internet Computing **28**(2), 57–64 (2024) https://doi.org/10.1109/MIC.2023.3344248

[27] Pearl, J.: Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference. San Mateo, Calif. : Morgan Kaufmann, ??? (1988)

[28] Pearl, J., Mackenzie, D.: The Book of Why: The New Science of Cause and Effect, 1st edn. Basic Books, Inc., USA (2018)

[29] Ganguly, N., Fazlija, D., Badar, M., Fisichella, M., Sikdar, S., Schrader, J., Wallat, J., Rudra, K., Koubarakis, M., Patro, G.K., Amri, W.Z.E., Nejdl, W.: A Review of the Role of Causality in Developing Trustworthy AI Systems. arXiv (2023). https://doi.org/10.48550/arXiv.2302.06975

[30] Tariq, M., Zeitoun, A., Valancius, V., Feamster, N., Ammar, M.: Answering what-if deployment and configuration questions with wise. ACM SIGCOMM Computer Communication Review (2008) https://doi.org/10.1145/1402946.1402971

[31] Hoffman, M.D., Johnson, M.J.: Elbo surgery: yet another way to carve up the variational evidence lower bound. In: Workshop in Advances in Approximate Bayesian Inference, NIPS, vol. 1 (2016)

[32] Huang, X., He, L., Zhang, W.: Vehicle Speed Aware Computing Task Offloading and Resource Allocation Based on Multi-Agent Reinforcement Learning in a Vehicular Edge Computing Network. In: IEEE International Conference on Edge Computing (EDGE) (2020). https://doi.org/10.1109/EDGE50951.2020.00008

[33] Guo, H., Liu, J., Lv, J.: Toward Intelligent Task Offloading at the Edge. IEEE

Network **34**(2), 128–134 (2020) https://doi.org/10.1109/MNET.001.1900200

[34] Sedlak, B., Casamayor Pujol, V., Donta, P.K., Dustdar, S.: Controlling Data Gravity and Data Friction: From Metrics to Multidimensional Elasticity Strategies. In: IEEE SSE 2023, Chicago, IL, USA (2023)

[35] Wang, J., Zhao, L., Liu, J., Kato, N.: Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. IEEE Transactions on emerging topics in computing **9**(3), 1529–1541 (2019)

[36] Tang, M., Wong, V.W.: Deep reinforcement learning for task offloading in mobile edge computing systems. IEEE Transactions on Mobile Computing **21**(6), 1985–1997 (2020)

[37] Huang, J., Wan, J., Lv, B., Ye, Q., Chen, Y.: Joint computation offloading and resource allocation for edge-cloud collaboration in internet of vehicles via deep reinforcement learning. IEEE Systems Journal **17**(2), 2500–2511 (2023)

[38] Ju, Y., Chen, Y., Cao, Z., Liu, L., Pei, Q., Xiao, M., Ota, K., Dong, M., Leung, V.C.: Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach. IEEE Transactions on Intelligent Transportation Systems **24**(5), 5555–5569 (2023)

[39] Levchuk, G., Pattipati, K., Serfaty, D., Fouse, A., McCormack, R.: Active Inference in Multiagent Systems: Context-Driven Collaboration and Decentralized Purpose-Driven Team Adaptation. In: Artificial Intelligence for the Internet of Everything. Academic Press, ??? (2019)

[40] Lu, S., Wu, J., Lu, P., Wang, N., Liu, H., Fang, J.: QoS-Aware Online Service Provisioning and Updating in Cost-Efficient Multi-Tenant Mobile Edge Computing. IEEE Services Computing (2023)

[41] Zhang, Z., Zhao, Y., Liu, J.: Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster. In: Service-Oriented Computing, Cham (2023). https://doi.org/10.1007/978-3-031-48424-7_18

[42] Sedlak, B., Pujol, V.C., Donta, P.K., Dustdar, S.: Equilibrium in the Computing Continuum through Active Inference. Future Generation Computer System (2024)

[43] Vilas, M.G., Auksztulewicz, R., Melloni, L.: Active Inference as a Computational Framework for Consciousness. Review of Philosophy and Psychology **13**(4), 859–878 (2022) https://doi.org/10.1007/s13164-021-00579-w

[44] Heins, C., Millidge, B., Demekas, D., Klein, B., Friston, K., Couzin, I., Tschantz, A.: pymdp: A Python library for active inference in discrete state spaces. Journal

of Open Source Software (2022)

[45] Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Run-time Adaptation of Data Stream Processing Systems: The State of the Art. ACM Computing Surveys (CSUR) (2021) https://doi.org/10.1145/3514496 . Publisher: ACM PUB27 New York, NY. Accessed 2022-04-28

[46] Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., Watson, M.: A buffer-based approach to rate adaptation: evidence from a large video streaming service. In: Proceedings of the 2014 ACM Conference on SIGCOMM. SIGCOMM '14, pp. 187–198. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2619239.2626296 . https://dl.acm.org/doi/10.1145/2619239.2626296 Accessed 2024-08-01

[47] Khani, M., Hamadanian, P., Nasr-Esfahany, A., Alizadeh, M.: Real-Time Video Inference on Edge Devices via Adaptive Model Streaming, pp. 4572–4582 (2021). https://openaccess.thecvf.com/content/ICCV2021/html/Khani_Real-Time_Video_Inference_on_Edge_Devices_via_Adaptive_Model_Streaming_ICCV_2021_paper.html Accessed 2024-08-01

[48] Ma, X., Su, Z., Xu, Q., Ying, B.: Edge Computing and UAV Swarm Cooperative Task Offloading in Vehicular Networks. In: 2022 International Wireless Communications and Mobile Computing (IWCMC), pp. 955–960 (2022)

[49] Cao, C., Dai, M., Shen, B., Zou, G., Dong, W.: Neural adaptive IoT streaming analytics with RL-Adapt. Computer Networks **235**, 109924 (2023) https://doi.org/10.1016/j.comnet.2023.109924 . Accessed 2024-08-01

[50] Yazdi, M., Khan, F., Abbassi, R., Quddus, N.: Resilience assessment of a subsea pipeline using dynamic Bayesian network. Journal of Pipeline Science and Engineering **2**(2), 100053 (2022) https://doi.org/10.1016/j.jpse.2022.100053

[51] Toğaçar, M.: Detecting attacks on IoT devices with probabilistic Bayesian neural networks and hunger games search optimization approaches. Transactions on Telecommunications Technologies (2022) https://doi.org/10.1002/ett.4418

[52] Kitson, N.K., Constantinou, A.C., Guo, Z., Liu, Y., Chobtham, K.: A survey of Bayesian Network structure learning. Artificial Intelligence Review **56**(8), 8721–8814 (2023) https://doi.org/10.1007/s10462-022-10351-w

[53] Dustdar, S., Pujol, V.C., Donta, P.K.: On Distributed Computing Continuum Systems. IEEE Transactions on Knowledge and Data Engineering **35**(4), 4092–4105 (2023) https://doi.org/10.1109/TKDE.2022.3142856

[54] Mengshoel, O., Poll, S., Kurtoglu, T.: Developing Large-Scale Bayesian Networks by Composition: Fault Diagnosis of Electrical Power Systems in Aircraft and Spacecraft (2009)

[55] Sedlak, B., Pujol, V.C., Donta, P.K., Dustdar, S.: Diffusing High-level SLO in Microservice Pipelines. In: 2024 IEEE International Conference on Service-Oriented System Engineering (SOSE) (2024)

[56] Chen, Y., Liu, S.: A novel learning method for feature evolvable streams. Evolving Systems (2024) https://doi.org/10.1007/s12530-024-09590-9

[57] Ankan, A., Textor, J.: pgmpy: A Python Toolkit for Bayesian Networks (2023)

[58] Yi, S., Hao, Z., Zhang, Q., Zhang, Q., Shi, W., Li, Q.: LAVEA: latency-aware video analytics on edge computing platform. In: Proceedings of the Second ACM/IEEE Symposium on Edge Computing. SEC '17, pp. 1–13. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132211.3134459

[59] Sedlak, B., Murturi, I., Donta, P.K., Dustdar, S.: A Privacy Enforcing Framework for Transforming Data Streams on the Edge. IEEE Transactions on Emerging Topics in Computing (2023) https://doi.org/10.1109/TETC.2023.3315131

[60] Varghese, R., M., S.: YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. In: ADICS (2024)

[61] Liu, Z., Li, Q., Chen, X., Wu, C., Ishihara, S., Li, J., Ji, Y.: Point Cloud Video Streaming: Challenges and Solutions. IEEE Network **35**(5), 202–209 (2021) https://doi.org/10.1109/MNET.101.2000364 . Conference Name: IEEE Network

[62] Dzung, N.M.: maudzung/SFA3D (2020). https://github.com/maudzung/SFA3D Accessed 2024-07-19

[63] Ahmadinia, A., Singh, A., Hamadani, K., Jiang, Y.: Tracking objects using QR codes and deep learning. In: Zhou, J.J., Osten, W., Nikolaev, D.P. (eds.) Fifteenth International Conference on Machine Vision (ICMV 2022). SPIE, Rome, Italy (2023). https://doi.org/10.1117/12.2679910

[64] opencv: opencv at 4.9.0 (2024). https://github.com/opencv/opencv/tree/4.9.0