Active Inference on the Edge: A Design Study*

Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology (*TU Wien*), Vienna 1040, Austria.
Email: {b.sedlak, v.casamayor, pdonta, dustdar}@dsg.tuwien.ac.at

***Abstract*—Machine Learning (ML) is a common tool to interpret and predict the behavior of distributed computing systems, e.g., to optimize the task distribution between devices. As more and more data is created by Internet of Things (IoT) devices, data processing and ML training are carried out by edge devices in close proximity. To ensure Quality of Service (QoS) throughout these operations, systems are supervised and dynamically adapted with the help of ML. However, as long as ML models are not retrained, they fail to capture gradual shifts in the variable distribution, leading to an inaccurate view of the system state. Moreover, as the prediction accuracy decreases, the reporting device should actively resolve uncertainties to improve the model's precision. Such a level of self-determination could be provided by Active Inference (ACI) – a concept from neuroscience that describes how the brain constantly predicts and evaluates sensory information to decrease long-term surprise. We encompassed these concepts in a single action-perception cycle, which we implemented for distributed agents in a smart manufacturing use case. As a result, we showed how our ACI agent was able to quickly and traceably solve an optimization problem while fulfilling QoS requirements.**

***Index Terms*—Active Inference, Machine Learning, Edge Intelligence, Service Level Objectives, Markov Blanket**

I. Introduction

Recent years have reported a constant transition of logic from the central cloud towards the edge of the network [1], thus, closer to the Internet of Things (IoT) devices that actually generate data. This transition includes the training of Machine Learning (ML) models (i.e., to save bandwidth and improve privacy), as well as data processing (i.e., to decrease latency) [2]. As soon as training has finished, ML models are a common measure to interpret and predict the behavior of distributed systems, e.g., to estimate the impact of redeployment [3] or forecast potential system failures [4], which must be circumvented to ensure the Quality of Service (QoS).

ML models are applied throughout the Computing Continuum (CC), i.e., from the cloud, over the fog, to the network edge – close to where models were trained. However, in many cases, ML models are not retrained, although new observations would be available [3], [4]; this inevitably leads to an inaccurate view of the system state, which, in turn, decreases the quality of any inference mechanism that uses the ML model. Imagine an elastic computing system, like envisioned in [5], [6], which observes the system through a set of metrics, evaluates whether QoS requirements – also called Service Level Objectives (SLOs) – were 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 the metrics correctly, and any consequential reconfiguration will fail to fulfill its purpose.

Ensuring the precision of an ML model requires a continuous feedback mechanism; such behavior could, for example, be achieved by optimizing a value function, as in reinforcement learning [7], [8]. However, we believe that this requires a more holistic approach, which starts with making the SLOs first-class citizens during the ML training process. Further, we want to highlight the responsibility of any service that uses the ML model to actively resolve or report ambiguities. Such a level of self-determination could be provided by Active Inference (ACI), a concept from neuroscience that describes how the brain constantly predicts and evaluates sensory information to decrease long-term surprise. ACI combines various concepts that have already been rudimentarily implemented in distributed systems, e.g., causal inference to identify dependencies between system parts [3], or dynamic adaptations of the system to ensure QoS – called homeostasis. This shows the potential of ACI.

In this paper, we advance one step further by combining the ACI concepts in a comprehensive design study of an ACI agent that optimizes the throughput of a smart factory. Internally, the agent follows an action-perception cycle: First, it estimates which parameter assignments would violate given SLOs, then it compares this expectation with new observations, and finally, it adjusts its beliefs (i.e., the ML model) accordingly. The agent focuses on exploring values that promise a high throughput while avoiding such that are likely to violate the SLOs. Furthermore, it favors solutions that are likely to improve the model precision, which, in turn, provides the agent with a clear understanding of the causal relations between model variables. Hence, the contributions of this article are the following:

- A novel ML paradigm based on ACI that continuously evaluates the quality of predictions. Thus, agents improve the model precision to ensure QoS for ongoing operations.
- The composite representation of an agent's behavior throughout the action-perception cycles. The distinct factors can be fine-tuned to determine the agent's preferences.
- A complete design study for a smart manufacturing agent that paves the way for other researchers to implement ACI in related automative use cases.

The remainder of the paper is structured as follows: Section II provides background information on ACI principles in distributed systems; in Section III we present existing work that included ACI; within Section IV we outline the design process of an ACI agent, which we implemented and evaluated in Section V. Finally, Section VI concludes the paper.

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

II. Background

We consider ACI an unknown concept for most readers outside of neuroscience; therefore, we use this section to summarize core concepts of ACI according to Friston et al. [8]–[12]. This includes but is not limited to (1) free energy minimization, (2) hierarchical organization of beliefs, (3) action-perception cycles, and (4) Bayesian inference and belief updating. Following that, we delineate our view of the intersection between ACI and distributed systems, in particular edge computing.

A. Active Inference Principles

To interpret observable processes, agents construct generative models, e.g., a person would reason that it rains due to water drops falling from the sky. Based on these observations, the agent can learn to understand real-world processes. However, if the generative model and the process diverge, the agent will eventually be "surprised", e.g., because water drops were caused by a neighbor watering her plants. The discrepancy (or uncertainty) between the agent's understanding of the process and reality is called Free Energy (FE); a more accurate understanding decreases FE at the same time.

More formally, the surprise $\Im(o|m)$ of observation o given model m is the negative log-likelihood of the observation. The surprise itself is capped by the FE of the model – expressed as the Kullback-Leibler divergence (D_{KL}) between approximate posterior probability (Q) of the hidden states (x) and their exact posterior probability (P). While mathematical approaches, such as exemplified in Eq. (1) & (2), provide a much-needed notation for working with the FE principle, in practice, many variables are computationally intractable, e.g., the true probability P.

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

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

Internally, agents organize their generative models in hierarchical structures; each level interprets lower-level causes and, based on that, provides predictions to higher levels. For example, suppose it rains with a certain probability, I bring an umbrella. This is commonly known as Bayesian inference and allows agents to use existing beliefs (widely known as priors) to calculate the probability of related events. Such decision processes can be segregated into self-contained causal structures (i.e., Markov blankets), e.g., one to interpret the weather and another to dress. As the agent infers that it is raining, he decides to pick the umbrella. However, when dressing, the agent only considers the weather (*rainy* or *sunny*) and disregards lower-level observations that led him to conclude that it's raining (e.g., humidity or obscurity).

ACI agents constantly engage in action-perception cycles, where they (1) predict sensory inputs, actively seek the information, and update their beliefs depending on the outcome – widely known as predictive coding. Afterward, they (2) can adjust the world to their existing beliefs through their own actions. While pragmatic actions (e.g., picking an umbrella) fulfill agents' preferences (e.g., staying dry), agents improve their decision-making by exploring the environment through epistemic actions. For example, a mere look at the sky reveals that the neighbor has watered her plants, avoiding surprise when wrongfully leaving with an umbrella. The agent thus updates its prior beliefs (i.e., rain $\rightarrow$ water) according to new data (i.e., rain $\rightarrow$ water $\leftarrow$ flowers) to form posterior beliefs.

B. ACI Principles in Distributed Systems

ACI encompasses multiple concepts; although there exist few implementations that combine them in one framework, most of them can be encountered in distributed systems. In the following, we review the principles described above and map them to existing concepts as far as possible:

1) Causal Inference: Causal structures (e.g., Bayesian networks [13]) can be trained to identify dependencies between parts of distributed systems. As pointed out by [14], causal structures have the fundamental advantage (over deep learning) of justifying their actions or recommendations, thus improving trustworthiness. Distributed systems can explain how metrics (e.g., latency or CPU load) are related to the system state [5], backtrack which service or device caused a system failure [3], or predict the impact of redeployment [15].

2) Free Energy Minimization: AI models are trained to improve their prediction accuracy, which, in turn, reduces their FE. Energy-based models [16], in particular, rate uncertainty as (free) energy. To ensure model accuracy over time, one option is to continuously report prediction errors (e.g., [17]). However, in many cases, systems lack adequate feedback loops and thus fail to capture gradual shifts in the variable distributions (e.g., [3], [15], [18]) While ML training is essential to decrease FE, epistemic actions often suffice to reduce uncertainties about expected outcomes: Distributed systems resolve contextual information before executing pragmatic actions, e.g., identify a low-utilized agent for load balancing or task offloading [19], [20], or evaluate resource availability before scaling a system [5], [21]. It is the general tradeoff between seeking either pragmatic value (exploitation) or information (exploration); multi-agent systems (e.g., [22]) control this through a hyperparameter called "exploration rate", which fosters early exploration of a global value space but decays over time as agents report little improvement. To improve generative models whenever feasible, this is also implemented for edge-based systems [19].

3) Homeostasis: The ultimate goal for an ACI agent is to persist over time; this requires maintaining certain internal variables under control. This concept is called homeostasis and can be found in various systems: The human body, for example, requires a core temperature of 37° for chemical processes; distributed systems, on the other hand, specify QoS requirements as SLOs [5], [6], [21], [23]. While the human body has its own temperature-controlling mechanisms, distributed systems rely on elasticity strategies to ensure QoS, e.g., by scaling computational resources to cap response time. Although surprise plays a significant role here, e.g., when reporting SLO violations, the preferred strategy is to engage with the environment to correct this instead of changing the perception.

III. Related Work

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

The authors in [24] discuss ACI as a general computational framework, highlighting how existing research used ACI for (simulating) sensory processing. Touching on the design of ACI agents, Heins et al. [25] provide a Python simulation that exemplifies how to structure action-perception cycles. Heins et al. further remark that existing ACI research largely focuses on formally constructing models in isolated environments such as Matlab SPM (e.g., [11]) rather than putting them into action, e.g., to improve the precision of ML models. A more hands-on application of ACI is thus to extend reinforcement learning with ACI principles [7], [8]. However, most research to date either uses only a few ACI principles or is not applied enough to easily transfer presented concepts to distributed systems.

The work in [22] is, therefore, an exception because it embeds ACI into the IoT and describes how ACI 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 exchange of experiences 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 ACI in distributed systems: causal inference and homeostasis. In this paper, we will present an agent that uses all three ACI principles to infer actions, maintain agents' internal equilibrium, and persist over time. Nevertheless, we will use the representation from [22] for FE minimization.

IV. Active Inference Design Process

In the following, we will walk through the design of an ACI agent by (1) building upon ACI background information to draw an action-perception loop, (2) describing a use case where the agent trains a model from scratch to optimize performance, (3) marking the boundaries of the generative model trained, and (4) defining the agent's behavior throughout the cycles.

A. Action-Perception Loop

To continuously ensure the precision of ML models and any consequential action, we will employ self-evidenced agents, i.e., they reason about their environment and train models autonomously. To that extent, ACI agents operate in action-perception cycles; each iteration aims to improve the accuracy of the model, infer optimal actions, and thus persist over time. As such, agents can be embedded into distributed systems, e.g., to maintain the QoS for a distributed task.

Fig. 1 provides a high-level overview of the steps that are repeated by the agent: Initially, a set of SLOs define the agent's preferences (e.g., $delay \leq \delta$) and establishes its expectations prior to evaluating any sensory data. The agent then assembles a causal graph to determine which factors influence these parameters; the conditional probability table contains the degree

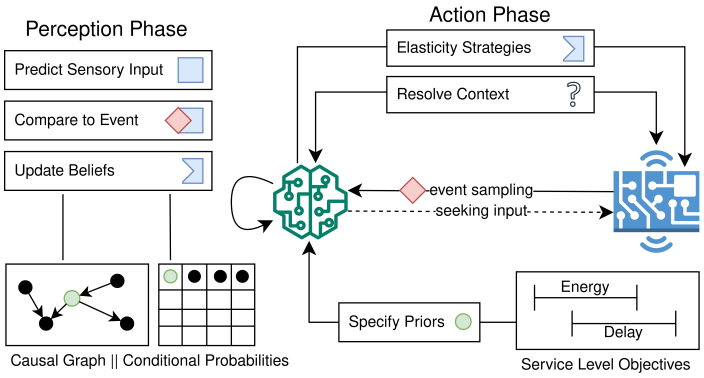


Fig. 1: Overview of the action-perception cycle in ACI

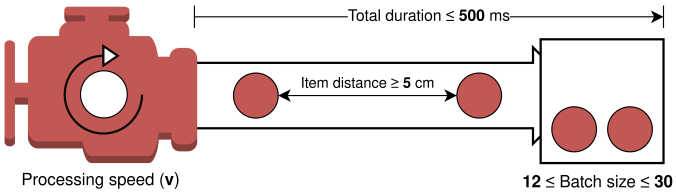


Fig. 2: A factory producing machine parts in batches

to which they are affected. Afterward, the agent starts to continuously predict the probability of observations, might actively seek a corresponding input, and then compares the event against the expectation. To decrease FE, the agent now has three options: (1) adjust its beliefs accordingly, i.e., update the causal graph and conditional probabilities; (2) change the environment toward its preferences, e.g., executing elasticity strategies; or (3) resolve contextual information to improve decision-making.

B. Use Case Description

The following use case is embedded in the smart manufacturing environment, which provides numerous opportunities for sensor-oriented analysis and dynamic adaptation of production. Fig. 2 provides a high-level overview of the use case:

Within a factory, machine parts are fabricated in batches of 12 to 30 pieces; a larger batch size increases the throughput and utilization of the factory engine. Each batch must be completed within 500 ms; thus, an increasing batch size decreases the timeframe for processing each part. The engine's utilization is supposed to impact the processing duration, though the magnitude is unknown. Also, due to a consecutive assembly step, the distance between parts should be above 5 cm. Given this setup, the factory manager would like to know the largest batch size that fulfills all constraints. However, because they lack historical data, it is difficult to answer this by training an ML model; this issue must be actively explored. Ideally, the learning process would also be autonomous and allow the factory to simultaneously produce their goods.

To provide the factory manager with the optimal batch size, we supervise the factory engine through an ACI agent. The resulting smart engine now enters what can be understood

TABLE I: Model variables and their boundaries

Name	Unit	Description	Range
batch size	num	number of machine parts per batch	$[12, 30]$
utilization	%	utilization of the factory engine	$[1, 100]$
distance	cm	space between two machine parts	$[1, \infty[$
part delay	ms	processing time per machine part	$[1, \infty[$
batch delay	ms	total time for batch processing	$[1, \infty[$

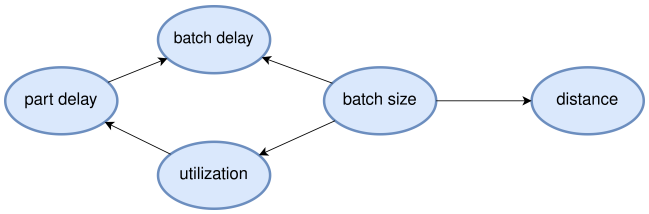


Fig. 3: Initial beliefs of relations between model variables

as a continuous calibration mechanism: throughout its action-perception cycle, it (1) estimates if an increase or decrease in batch size would violate the given constraints (i.e., its SLOs), (2) compares the expectation with the result, and (3) continuously explores the value space by slightly varying the batch size. The agent thus gradually approaches solutions that promise high throughput while satisfying all constraints.

C. Generative Model Setup

While the use case showed how ACI can help solve optimization problems, we will now dive deeper into the generative model created by the agent. The design process is loosely oriented towards the guidance provided by Parr et al. [26], which depicts an abstract sequence of steps to design ACI systems. The main questions we aim to answer are:

1) What is part of the generative model, and what are the interfaces to the exterior?
2) What is the hierarchical and temporal depth of the model, and how do they affect causal inference?
3) What are the model variables and prior beliefs – what can be modified (i.e., learned), and what is immutable?

To predict whether a *batch size* would fulfill the SLOs, agents must identify the variables that have an impact on them. These could be extracted through a causal structure (e.g., [3], [15]) or, in the absence of training data, come from expert knowledge, which can be updated over time. The manager initially believes that variables are related as depicted by the Directed Acyclic Graph (DAG) in Fig. 3; the respective variables are described in Table I. Variables in ACI represent an interface between the generative model and the exterior world, i.e., if the utilization of the physical engine changes, this is reflected through the respective variable (i.e., *utilization*), which in turn determines the internal view of the system state. Information provided through interface variables is used to construct the generative model, but also to evaluate SLO fulfillment (e.g., *batch delay* $\leq$ 500 ms). To that extent, it needs to analyze the respective variable (from the DAG), as well as its parent, child, and spouse nodes. This subset provides a causal filter to the variable state, called Markov blanket. A central premise is that all these sensory variables accurately reflect the exterior; otherwise, subsequent decisions (e.g., decreasing *batch size* to decrease *delay*) would perpetuate any measurement error.

For the given use case, we use an SLO-induced boundary as our natural limit on temporal depth: Equal to the maximum *batch delay (bd)*, each action-perception cycle lasts 500 ms. Within each cycle, the agent predicts the engine's behavior (i.e., reflected through the metrics) over the next 500 ms; afterward, the prediction is compared against the events observed during that time. While the cycle's length can be chosen freely, longer periods decrease the prediction accuracy or increase the computational complexity (i.e., to evaluate the SLO once, it must consider multiple cycles or fractions of them). The hierarchical depth, on the other hand, is determined by the number of variables and edges in the model. A deeper hierarchy would increase the complexity of model training and inference; however, the use case does not provide variables other than the ones already contained in the DAG.

So far, it only remains to explain what priors are in the given example: Priors are our assumptions about the system before verifying them, e.g., which *batch size* should provide the highest throughput without violating the SLOs. Priors are subject to the learning process, while SLOs are fixed; each action-perception cycle aims to improve the generative model's accuracy, thus decreasing FE. As we will see in Section V, the initial beliefs (i.e., before evaluating any cycle) speed up the convergence of ACI to the optimal solution.

D. Active Inference Agent

To find the optimal *batch size*, the central mechanism of the agent is the action-perception cycle shown in Fig. 1. Initially, the agent has little information available to form priors or infer a fitting *batch size*; however, as the agent samples the environment through its interface variables, each cycle adds new observations (s_n) to the total amount of known samples (s_k). The agent's behavior throughout each cycle (i.e., how it interprets sensory information and which action it takes) is determined by three main factors: (1) pragmatic value (pv) of actions; (2) ambiguity or risk assigned (ra) to actions; and (3) epistemic value or information gained (ig) by actions. The following notation of these factors is related to [22], [26], though the composition is different. The only parameter that the agent can actively set is *batch size*; the remaining variables are causally influenced by this factor. Thus, if the agent changes the *batch size*, this is reflected through the related variables.

The pragmatic value that emerges from higher *batch size* is simple: more throughput. Therefore, we define $pv(bs) = bs \times \frac{100}{30}$, which encourages the agent to increase *batch size*. The multiplier $\frac{100}{30}$ scales the factor to the range $[1, 100]$, which is equal for all three factors. Contrarily, high *batch size* might exceed $bd \leq 500ms$ or $d \geq 5cm$, i.e., the SLOs associated with *batch delay* and *distance* (d). To evaluate the risk of violating the SLOs we consider how often past observations for a *batch size* (s_{kb}) have violated the SLOs. The ra, e.g., for $batchsize = 20$, would thus be determined by the rate between samples that

fulfilled the SLOs and the total number of samples ($|s_{kb}|$); this is formalized in Eq. (3) & (5). As long as the list of samples for a $batchsize$ (or short bs) is empty (i.e., $|s_{kb}| = 0$), the agent interpolates the value with the prior and latter ra as reference points, e.g., if the agent knows $ra(30) = 90$ and $ra(20) = 20$, in the absence of samples for $batchsize = 25$, it infers that $ra(25) = 55$. This interpolation is contained in Eq. (4).

$$ra(bs) = 100 - \begin{cases} inter(bs), & \text{if } |s_{kb}| = 0 \\ \frac{valid(bs)}{|s_{kb}|} \times 100, & \text{otherwise} \end{cases} \quad (3)$$

$$inter(bs) = ra_{i-1} + (bs - bs_{i-1}) \times \frac{(ra_{i+1} - ra_{i-1})}{(bs_{i+1} - bs_{i-1})} \quad (4)$$

$$valid(bs) = \sum_{i=1}^{|s_{kb}|} [(bd_i \leq 500) \wedge (d_i \geq 5)] \quad (5)$$

The ig of an action is determined by the ambiguity that it resolves; in other words, we aim to make future predictions less surprising. Reviving the idea of surprise from Eq. (1), we now require the surprise for s_n given s_k: Eq. (7) shows how the total surprise is the sum of surprises of new samples; $f(x)$ describes the probability density function[1] with $\mu = \bar{s_k}$ and σ_{s_k}. For each s_n, the surprise is appended to a list of past surprises $S = S \cup surprise(s_n, s_k)$; $S_x \in S$ contains all values with $x = batchsize$. If a $batchsize$ has reported repeatedly surprising values, it supposedly provides more information gain: This is reflected through Eq. (6) because the median surprise ($\tilde{S}_x$) will rise above the global average ($\bar{S}$). To foster exploration of prior unknown $batchsize$, in the absence of surprise values, e.g., $|S_{25}| = 0$, it assumes $ra(25) = \max(S)$.

$$ig(bs) = \left(\frac{\tilde{S}_{bs}}{\bar{S}} \right)_{|s_k|} \times 100 \quad (6)$$

$$surprise(s_n, s_k) = \sum_{i=1} -\log f(d_i) \quad (7)$$

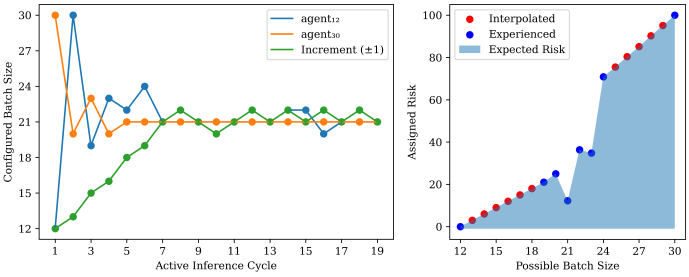
Ultimately, to evaluate the potentials but also risks that emerge from each $batchsize$, the three factors are merged into a common one – (cf). Since all factors are scaled to the range $[1, 100]$, they can be combined as $cf(bs) = pv(bs) - ra(bs) + ig(bs)$. At the end of each cycle, the agent resolves $cf(x)$ for $x = [12, 30]$ and chooses the highest scoring as new $batchsize$.

V. Evaluation

To evaluate the ideas presented in the last Section, we provide a Python implementation of the ACI agent that comprises the action-perception loop to create a generative model. Although we did not embed the agent in a physical engine to measure sensory information, we used a compatible data set generated with [27] to simulate an equal behavior. The prototype of the agent, the data, as well as the analysis are available on GitHub[2]. The agent starts the simulation by processing a batch of items

[1]A function that described the likelihood of an observation o in a continuous range given that the probabilities are distributed with $O \sim \mathcal{N}(\mu, \sigma)$.

[2]https://anonymous.4open.science/r/analysis-20F6/DATE/



(a) Batch size / cycle (b) Risk / batch size

Fig. 4: History of best scoring *batch size* and associated risk

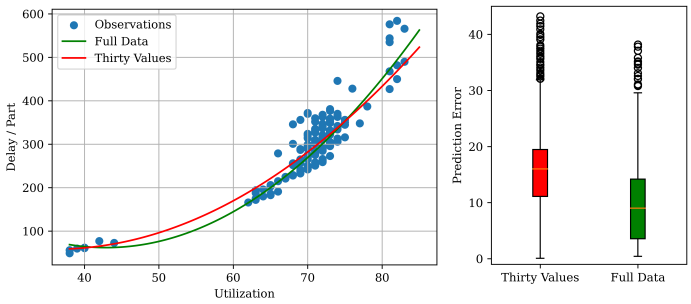
and observes for each item a set of metrics, which represent the variables from Table I. In each round, the agent computes the factors that determine its behavior (i.e., pv, ra, and ig) as described, chooses the highest common factor (cf), and instructs the engine to operate with the new $batchsize$. This concludes one iteration in the action-perception cycle.

A. Comparative Analysis & Results

We evaluated two main aspects of the implementation: (1) which *batch size* it chooses at the end of each cycle, and (2) how well the generative model can reflect the partially observable relation between *utilization* and *part delay*.

Fig. 4a tracks the chosen *batch size* depending on the cf score: The blue line depicts the agent's behavior when starting with $batchsize = 12$, and the red line when starting with 30, i.e., the safest or most ambitious priors. $Agent_{30}$ reaches a *batch size* of 21 after 5 iterations; whether this is a good (or optimal) solution is determined by multiple opposite factors: As *batch size* increases, both pa and ra rise. To provide more detail, Fig. 4b contains the ra that $agent_{30}$ assigned to each $batchsize$ after 100 iterations. Operating with $batchsize = 21$, $agent_{30}$ reported SLO violations for 12% of all observations. If this cannot be tolerated, the ra must be adjusted accordingly; otherwise, 21 presents a very high (if not optimal) pv because any larger *batch size* would be more than three times more likely to violate the SLOs, according to their ra. Complementarily, the green line shows the agent's behavior if it simply increases or decreases the *batch size* depending on whether SLOs were fulfilled for the current batch.

While one goal was to reach a high pv, the agent's intrinsic motivation is to decrease the FE by developing an accurate generative model. This includes estimating the magnitude of causal relations such as *utilization* $\rightarrow$ *part delay*. Therefore, after receiving a number of samples, the agent can use (polynomial) regression to infer *part delay* for unknown *utilization*. Fig. 5a shows a 2D representation of this relation for 2500 processed batch items, supervised by $agent_{12}$; the red line represents the agent's internal model after training on all observations, and the red line after training on only 30 values. After the first round, $agent_{12}$ decided to explore only $batchsize \geq 19$, thus, Fig. 5a contains no observations for *utilization* $[45, 60]$. The distribution of prediction errors between the regression functions and all items is shown in Fig. 5b. We observe that



(a) Polynomial regression function (b) Prediction errors

Fig. 5: Estimated relation between *utilization* and *part delay*

a larger sample size improved the accuracy, but also that a relatively small number of samples (i.e., 30) provided initially acceptable results.

VI. Conclusion

An ML model should not only reflect a generative process at one moment; ideally, it should persist over time without losing its precision. However, as long as the model is not retrained, a lack of continuous feedback inevitably leads to poor accuracy. Thus, any system that aims to dynamically adapt its service must supervise its inference mechanisms to ensure QoS is fulfilled. With the intention to solve this, we presented a distributed agent that is based on Active Inference – a concept from neuroscience that describes how the brain constantly predicts and evaluates sensory information to decrease long-term surprise. Operating in cycles, the agent maximizes the model evidence by exploring the space of values that fulfill the QoS. Thus, the agent improves any decision-making based on the ML model because ambiguities are repeatedly resolved.

To solve a smart manufacturing use case, we presented a design study that defines the agent's behavior when creating a generative model. Which action the agent takes and how it adapts its beliefs is determined by three main factors: pragmatic value, assigned risk (of violating SLOs), and information gain. We implemented the ACI agent in Python and tracked each cycle's preferred action – including the factors that led to it – and the agent's causal understanding between two variables. After 5 cycles, the agent converged to a solution that presented an optimal tradeoff between high pragmatic value and negligible SLO violations. Further, the agent needed only 30 observations (i.e., 2 cycles) to estimate a previously unknown variable relation. Exploring causalities between variables and constructing the agent's behavior from empirical factors makes the produced solutions traceable. Based on these results, we see a strong potential for ACI to support elastic computing systems by continuously ensuring the precision of ML models.

References

[1] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence," *IEEE Internet of Things Journal*, Aug. 2020.

[2] V. C. Pujol, P. K. Donta, A. Morichetta, I. Murturi, and S. Dustdar, "Edge intelligence—research opportunities for distributed computing continuum systems," *IEEE Internet Computing*, vol. 27, no. 4, pp. 53–74, 2023.

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

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

[5] B. Sedlak, V. Casamayor Pujol, P. K. Donta, and S. Dustdar, "Controlling Data Gravity and Data Friction: From Metrics to Multidimensional Elasticity Strategies," in *IEEE SSE 2023*, Chicago, IL, USA, Jul. 2023.

[6] S. Nastic, A. Morichetta, T. Pusztai, S. Dustdar, X. Ding, D. Vij, and Y. Xiong, "SLOC: Service Level Objectives for Next Generation Cloud Computing," *IEEE Internet Computing*, vol. 24, no. 3, May 2020.

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

[8] K. J. Friston, J. Daunizeau, and S. J. Kiebel, "Reinforcement Learning or Active Inference?" *PLOS ONE*, vol. 4, no. 7, p. e6421, Jul. 2009.

[9] K. Friston, "Life as we know it," *Journal of The Royal Society Interface*, vol. 10, no. 86, p. 20130475, Sep. 2013.

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

[11] R. Smith, K. J. Friston, and C. J. Whyte, "A step-by-step tutorial on active inference and its application to empirical data," *Journal of Mathematical Psychology*, vol. 107, p. 102632, Apr. 2022.

[12] N. Sajid, P. J. Ball, T. Parr, and K. J. Friston, "Active inference: demystified and compared," *Neural Computation*, vol. 33, no. 3, pp. 674–712, Mar. 2021.

[13] J. Pearl, *Probabilistic reasoning in intelligent systems : networks of plausible inference*. San Mateo, Calif. : Morgan Kaufmann, 1988.

[14] N. Ganguly *et al.*, "A Review of the Role of Causality in Developing Trustworthy AI Systems," Feb. 2023.

[15] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering what-if deployment and configuration questions with wise," *ACM SIGCOMM Computer Communication Review*, 2008.

[16] Y. W. Teh, M. Welling, S. Osindero, and G. E. Hinton, "Energy-Based Models for Sparse Overcomplete Representations," *Journal of Machine Learning Research*, vol. 4, pp. 1235–1260, Dec. 2003.

[17] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li, "AI^2: Training a Big Data Machine to Defend," in *IEEE Big Data Security*, Apr. 2016, pp. 49–54.

[18] M. Simsek, B. Kantarci, and Y. Zhang, "Detecting Fake Mobile Crowdsensing Tasks: Ensemble Methods Under Limited Data," *IEEE Vehicular Technology Magazine*, vol. 15, no. 3, pp. 86–94, Sep. 2020.

[19] X. Huang, L. He, and W. Zhang, "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)*, Oct. 2020.

[20] H. Guo, J. Liu, and J. Lv, "Toward Intelligent Task Offloading at the Edge," *IEEE Network*, vol. 34, no. 2, pp. 128–134, Mar. 2020.

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

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

[23] T. Pusztai, A. Morichetta, V. C. Pujol, S. Dustdar, S. Nastic, X. Ding, D. Vij, and Y. Xiong, "A Novel Middleware for Efficiently Implementing Complex Cloud-Native SLOs," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, Sep. 2021, pp. 410–420.

[24] M. G. Vilas, R. Auksztulewicz, and L. Melloni, "Active Inference as a Computational Framework for Consciousness," *Review of Philosophy and Psychology*, vol. 13, no. 4, pp. 859–878, Dec. 2022.

[25] C. Heins, B. Millidge, D. Demekas, B. Klein, K. Friston, I. Couzin, and A. Tschantz, "pymdp: A Python library for active inference in discrete state spaces," *Journal of Open Source Software*, May 2022.

[26] T. Parr, G. Pezzulo, and K. J. Friston, *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. The MIT Press, Mar. 2022.

[27] B. Sedlak, I. Murturi, P. K. Donta, and S. Dustdar, "A Privacy Enforcing Framework for Transforming Data Streams on the Edge," *IEEE Transactions on Emerging Topics in Computing*, 2023.