Anomaly Detection via Learning-Based Sequential Controlled Sensing

Joseph, Geethu; Zhong, Chen; Gursoy, M. Cenk; Velipasalar, Senem; Varshney, Pramod K.

DOI
10.1109/JSEN.2024.3399456

Publication date
2024

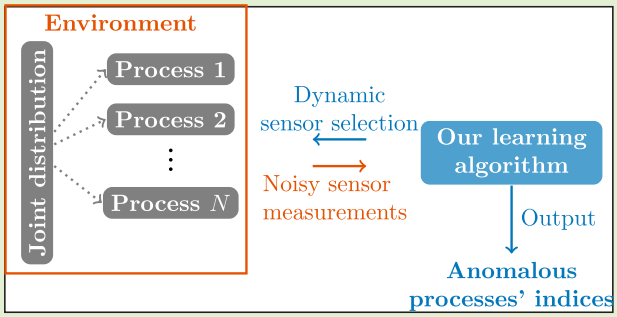
Document Version
Final published version

Published in
IEEE Sensors Journal

Anomaly Detection via Learning-Based Sequential Controlled Sensing

Geethu Joseph, *Member, IEEE*, Chen Zhong, M. Cenk Gursoy, *Senior Member, IEEE*, Senem Velipasalar, *Senior Member, IEEE*, and Pramod K. Varshney, *Life Fellow, IEEE*

***Abstract*—We tackle the anomaly detection problem within a given set of binary processes through a learning-based controlled sensing approach. This problem is particularly pertinent to applications related to the Internet of Things (IoT) that monitor multiple related processes. Each process is defined by a binary random variable indicating its anomalous status. To pinpoint anomalies, a decision-making agent observes a subset of processes at each time point, with each observation incurring an associated cost. We formulate a sequential selection policy that dynamically determines which processes to observe at each moment, aiming to minimize both decision delay and sensing cost. The conventional solution using model-based active hypothesis testing algorithms overlooks the joint statistics of the processes and fails to consider unequal sensing costs or errors in observations. To solve this problem, we, for the first time, pose it as a sequential hypothesis testing problem within the framework of Markov decision processes (MDPs), leveraging both a Bayesian log-likelihood ratio-based reward and an entropy-based reward. We address this problem through two approaches: 1) a deep reinforcement learning-based method involving the design of both deep *Q*-learning (DQN) and policy gradient actor–critic (AC) algorithms and 2) a deep active inference (AI)-based approach. Our model-based posterior updates to tackle the uncertainties in the observations, combined with the data-driven neural networks to handle the underlying statistical dependence between the processes, strike a balance between the model-based and the data-driven approaches. Our numerical experiments showcase the effectiveness of our algorithms, illustrating their capability to adapt to any unknown statistical dependence among the processes.**



***Index Terms*— Active hypothesis testing, active inference (AI), anomaly detection, quickest state estimation, sequential decision-making, sequential sensing.**

I. Introduction

SEQUENTIAL controlled sensing refers to a stochastic framework in which an agent sequentially controls the process of acquiring observations. The goal here is to minimize the cost of making observations while satisfying the inference objectives. We consider a sequential controlled sensing problem in the context of anomaly detection wherein there are N processes, each of which can be in either a normal or an anomalous condition. Our goal is to identify the anomalies among the given processes. To this end, the decision-making agent sequentially chooses a subset of processes (or equivalently sensors monitoring these processes) at every time instant, probes them, and obtains estimates of their conditions. The agent generally obtains noisy observations, i.e., the observed condition may get flipped from the actual condition with a certain probability. This paradigm is encountered in many practical applications such as remote health monitoring, assembly lines, structural health monitoring, and Internet of Things (IoT). In such applications, the objective is to identify the anomalies among a given set of different (not necessarily independent) functionalities of a system [1], [2]. Each sensor monitors a different functionality and sends observations to the agent over a communication link. The received observation may be distorted due to the unreliable nature of the sensor hardware and/or the noisy link (e.g., a wireless channel) between the sensor and the agent. Hence, the agent needs to probe each process multiple times before declaring one or more of the processes anomalous with the desired confidence.

Manuscript received 2 March 2024; accepted 6 May 2024. Date of publication 21 May 2024; date of current version 1 July 2024. This work was supported in part by the National Science Foundation under Grant ENG 60064237. An earlier version of this paper was presented in part at the IEEE International Workshop on Signal Processing Advances in Wireless Communications, May 2020, Atlanta, GA, USA [DOI: 10.1109/SPAWC48557.2020.9154275], and in part at the IEEE Global Communications Conference, December 2020, Taipei, Taiwan [DOI: 10.1109/GLOBECOM42002.2020.9322390]. The associate editor coordinating the review of this article and approving it for publication was Prof. Pierluigi Salvo Rossi. *(Corresponding author: Geethu Joseph.)*

Geethu Joseph is with the Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft Technical University, 2628 XE, Delft, The Netherlands (e-mail: g.joseph@tudelft.nl).

Chen Zhong, M. Cenk Gursoy, Senem Velipasalar, and Pramod K. Varshney are with the Department of Electrical and Computer Engineering, Syracuse University, New York, NY 13244 USA (e-mail: czhong03@syr.edu; mcgursoy@syr.edu; svelipas@syr.edu; varshney@syr.edu).

Digital Object Identifier 10.1109/JSEN.2024.3399456

Repeatedly probing all the processes allows the agent to find any potential system malfunction or anomaly quickly, but this incurs a higher energy consumption that reduces the life span of the network. Therefore, we address the question of how the agent must sequentially choose the subset of processes to accurately detect the anomalies while minimizing the delay and cost of making observations.

A. Literature Review

A classical approach to solve the sequential process selection problem for anomaly detection is based on the active hypothesis testing framework [3], [4]. Here, the decision-making agent constructs a hypothesis corresponding to each of the possible conditions of the processes and determines which one of these hypotheses is true. The goal of active hypothesis testing is to infer the true hypothesis by collecting relevant data sequentially until sufficiently strong evidence is gathered. In [5], Chernoff proposed a randomized strategy and established its asymptotic optimality. This seminal work was followed by several other studies that investigated active hypothesis testing under different settings. For example, in [6], a heuristic policy for controlled sensing using the extrinsic Jensen–Shannon divergence metric was introduced. Additionally, Naghshvar et al. [7] derived lower bounds characterizing fundamental limits on optimal reliability and upper bounds based on two heuristic policies for dynamically selecting actions. The Chernoff test was also applied to radar models [8] and extended to heterogeneous processes with varying prior distributions [9].

The studies above investigated the model-based algorithms that are designed under simplified modeling assumptions. More recently, researchers have turned to designing data-driven deep learning algorithms for active hypothesis testing. Notably, these data-driven approaches have demonstrated superiority over model-driven Chernoff-based methods and their variants [10] in the general active hypothesis testing problem. Later, the actor–critic (AC) deep reinforcement learning approach [3] has been applied to solve the controlled sensing problem, but the underlying statistical model was not fully leveraged. Our recent preliminary works have utilized the statistics of the underlying process to formulate a Markov decision process (MDP) and systematically applied deep learning techniques such as AC [4] and active inference (AI) algorithms [11] to the problem. These algorithms are not only model-free, offering increased flexibility compared to traditional methods, but also entail reduced computational complexity. Building upon these works, we conduct a more comprehensive and unified analysis of deep learning-based anomaly detection, rigorously establishing the feasibility of deep learning techniques for the controlled sensing problem.

We note that our anomaly detection problem is different from sequential parameter estimation. Sequential parameter estimation refers to estimating the random parameter of a process [12], [13], [14]. Although this goal is similar to ours, the controlled sequential selection of processes makes our problem fundamentally different from sequential parameter estimation. In particular, only one Bernoulli process is considered in [12], [13], and [14], and at every time instant, the decision-maker only decides whether or not to continue collecting observations. Hence, the results in these studies apply to our setting only if we consider the set of all N processes as a single random process with 2^N states and choose the suboptimal strategy of observing all the processes all the time.

B. Our Contributions

To the best of our knowledge, ours is the first work that formulated the anomaly detection problem as an active hypothesis testing problem and developed specific solutions for the problem. The goal of our paper is to explore the feasibility of deep learning approaches, such as Q-learning, AC, and AI by tailoring their design and architecture in the considered setting with the judicious choice of the state and action spaces and reward functions. We present a comparative analysis of their performances when applied to our specific problem, employing various metrics such as detection accuracy, sensing time, and cost as well as training overhead. Our specific contributions are as follows.

1) Formulation of Anomaly Detection as a Markov Decision Process: In Section III, we formulate the anomaly detection problem as an MDP. We define the posterior belief vector on the conditions of the processes as the state of the MDP, the subset of processes chosen by the decision-making agent as the action, and two different types of reward functions: an average Bayesian log-likelihood ratio (LLR)-based reward and an entropy-based reward. The rewards are designed such that the optimal policy of the MDP minimizes the sensing cost and the delay in decision-making. Finally, the process/sensor selection is formulated as the problem of finding an optimal policy that maximizes the long-term reward of the MDP subject to the condition that the confidence level on the estimate exceeds a specific value.

2) RL Algorithms: In Section IV, we develop the deep RL framework through which an optimal policy that maximizes the discounted cumulative reward is learned. We develop two deep RL algorithms, based on the Q-learning and AC frameworks.

3) Active Inference: In Section V, we present an alternative solution strategy called AI. Here, we define the notion of *free-energy* based on the entropy associated with the estimate of the states of the processes and the sensing cost and reformulate the anomaly detection problem as an AI problem to minimize the free energy. The resulting algorithm is implemented using deep neural networks which are relatively less explored for AI.

4) Empirical Validation: Via our numerical results presented in Section VI, we investigate the performance of different frameworks and algorithms in terms of detection accuracy, delay, and sensing cost. We show that the AI algorithm is more robust to the variations in the system parameters and adapts better to statistical dependence among the processes. Further, we observe that as the statistical dependence between the states of the processes increases, the delay in state estimation gets diminished. This result implies that unlike the traditional Chernoff test, our algorithms are able to learn and exploit any underlying statistical dependence among the processes to reduce the number of observations.

In summary, we use the model-based posterior updates to tackle the uncertainties in the observations and the data-driven neural networks to handle the underlying statistical dependence between the processes, balancing the model-based and the data-driven approaches.

Furthermore, in this article, compared to the conference versions [11], [15], we conduct a more comprehensive and unified analysis of deep learning-based anomaly detection and make several new contributions.

1) We design a new RL algorithm based on the deep Q-learning (DQN) algorithm which we implement using the dueling architecture.
2) In addition to the LLR-based reward, we introduce an entropy-based reward (newly applied in deep RL algorithms), and we mathematically show that the two reward functions encourage the agent to achieve the desired confidence level as quickly as possible (see Proposition 1).
3) We derive the Chernoff test for the anomaly detection problem and compare its performance with our algorithms.
4) We present a detailed numerical study that compares the different algorithms when the cost and flipping probabilities are different across the processes.

The remainder of the article is organized as follows. We present the system model in Section II and describe the MDP problem in Section III. In Sections IV and V, we present our RL and AI algorithms, respectively. We provide the simulation results in Section VI and offer our concluding remarks in Section VII.

II. Anomaly Detection Problem

We consider a set of N processes wherein each process is in one of the two conditions: normal (denoted by 0) or anomalous (denoted by 1). The condition of the ith process is denoted by the ith entry $\boldsymbol{x}_i$ of a random vector $\boldsymbol{x} \in \{0,1\}^N$. The vector $\boldsymbol{x}$ can take $M \triangleq 2^N$ possible values denoted by $\{\boldsymbol{h}^{(i)},\ i = 1, 2, \ldots, M\}$. The conditions of these processes (entries of $\boldsymbol{x}$) can be potentially statistically dependent. This dependence is captured by the prior distribution of $\boldsymbol{x}$ that is denoted using $\pi(0) \in [0,1]^M$ whose ith entry $\pi_i(0)$ is

$$\pi_i(0) = \mathbb{P}\{\boldsymbol{x} = \boldsymbol{h}^{(i)}\}. \tag{1}$$

Our goal is to identify the anomalous processes out of the N processes, which is equivalent to estimating the random vector $\boldsymbol{x}$. To estimate $\boldsymbol{x}$, the decision-making agent probes one or more processes at every time instant and obtains potentially erroneous observations of the corresponding entries of $\boldsymbol{x}$. Let the set of processes probed at time t be $\mathcal{A}(t) \in \mathcal{P}(N)$ where $\mathcal{P}(N)$ denotes the power set of $\{1, 2, \ldots, N\}$ without the null set (i.e., $|\mathcal{P}(N)| = M - 1$). Also, let the observation corresponding to the ith process at time t be denoted as $\boldsymbol{y}_i(t)$. Depending on the condition, $\boldsymbol{y}_i(t)$ obeys the following probabilistic model:

$$\boldsymbol{y}_i(t) = \begin{cases} \boldsymbol{x}_i, & \text{with probability } 1 - p_i \\ 1 - \boldsymbol{x}_i, & \text{with probability } p_i \end{cases} \tag{2}$$

where $p_i \in [0, 0.5]$ is called the crossover (or flipping) probability of the ith process. We also assume that given $\boldsymbol{x}$, the observations are jointly (conditionally) independent

$$\mathbb{P}\big[\{\boldsymbol{y}(\tau)\}_{\tau=1}^{t} \big| \boldsymbol{x}\big] = \prod_{\tau=1}^{t} \prod_{k=1}^{N} \mathbb{P}\big[\boldsymbol{y}_k(\tau) \big| \boldsymbol{x}\big] \tag{3}$$

for any integer $t > 0$ where $\boldsymbol{y}(\tau) \in \{0,1\}^N$ and its kth entry is $\boldsymbol{y}_k(\tau)$. Further, probing the ith process incurs a cost of $c_i > 0$. In short, at every time instant t, the decision-maker probes the processes indexed by $\mathcal{A}(t)$, obtains the corresponding observations denoted by $\boldsymbol{y}_{\mathcal{A}(t)}(t) \in \{0,1\}^{|\mathcal{A}(t)|}$, and incurs a sensing cost of $\sum_{k \in \mathcal{A}(t)} c_k$.

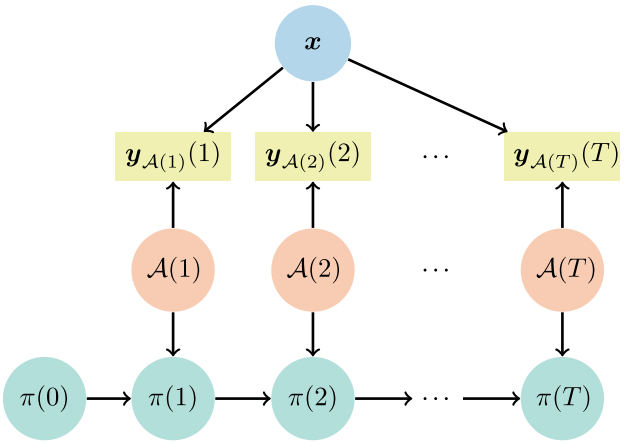


Fig. 1. Graphical model with posteriors (MDP states), actions, and observations.

In this setting, the three performance metrics associated with decision-making are the following.

1) *Stopping time* denoted by T which is the time instant when the decision-maker ends the observation acquisition phase and yields its estimate $\hat{\boldsymbol{x}}$ of $\boldsymbol{x}$.
2) *Detection accuracy* given by the conditional probability $\mathbb{P}\Big[\hat{\boldsymbol{x}} = \boldsymbol{x} \Big| \{\boldsymbol{y}_{\mathcal{A}(t)}(t)\}_{t=1}^{T}\Big]$.
3) *Sensing cost* given by $\sum_{t=1}^{T} \sum_{k \in \mathcal{A}(t)} c_k$ which represents the total cost incurred during the observation acquisition.

The strategy of probing all the processes at all times may lead to the most accurate and fastest decision, but at the expense of a higher sensing cost. Therefore, the decision-maker sequentially chooses a subset of processes $\mathcal{A}(t) \in \mathcal{P}(N)$ to balance the trade-offs between the three performance metrics. Our decision-making algorithm has two components.

1) *Sequential process selection:* A mechanism to choose $\mathcal{A}(t) \in \mathcal{P}(N)$ at every time t,
2) *Stopping rule:* A mechanism to determine when to stop taking observations and declare $\hat{\boldsymbol{x}}$.

We next present a novel anomaly detection algorithm that we derive by casting the detection as a learning problem using an MDP framework.

III. MDP-Based Learning Problem Formulation

This section describes the MDP framework that models the anomaly detection problem. An MDP represents a sequential decision-making problem in stochastic environments wherein the state of the environment depends on the action of a decision-maker. The goal of the decision-maker is to sequentially decide which action to choose while in a given state to

maximize the reward which is the same as finding a mapping from the states to the actions. This mapping is referred to as a policy and it can be either deterministic (i.e., a one-to-one mapping) or stochastic (described by conditional probability distributions over actions given the states).

A. State and Action

In the context of our anomaly detection problem, we define the state of the MDP at time t as the posterior belief vector $\pi(t)$ on the random vector $\boldsymbol{x} \in \{0,1\}^N$. The ith entry of the posterior belief vector $\pi(t) \in [0,1]^M$ is defined as

$$\pi_i(t) = \mathbb{P}\Big[\boldsymbol{x} = \boldsymbol{h}^{(i)} \Big| \{\boldsymbol{y}_{\mathcal{A}(\tau)}(\tau)\}_{\tau=1}^{t}\Big]. \tag{4}$$

Further, the action at time t refers to the set of processes to be observed, $\mathcal{A}(t) \in \mathcal{P}(N)$.

We next establish the connections between the states, actions, and corresponding observations which can be represented using a probabilistic graphical model depicted in Fig. 1. We first note that at time t, the data available to the agent is $\{\boldsymbol{y}_{\mathcal{A}(\tau)}(\tau), \tau = 1, 2, \ldots, k\}$ using which the posterior belief vector $\pi(t) \in [0,1]^M$ can be computed in closed form. From (4), $\pi_i(t)$ is computed recursively from (3) and (1) as

$$\pi_i(t) = \frac{\pi_i(t-1) \prod_{k\in\mathcal{A}(t)} \mathbb{P}\big[\boldsymbol{y}_k(t) \big| \boldsymbol{x} = \boldsymbol{h}^{(i)}\big]}{\sum_{j=1}^{M} \pi_j(t-1) \prod_{k\in\mathcal{A}(t)} \mathbb{P}\big[\boldsymbol{y}_k(t) \big| \boldsymbol{x} = \boldsymbol{h}^{(j)}\big]} \tag{5}$$

where we obtain from (2) that

$$\mathbb{P}\big[\boldsymbol{y}_k(t) \big| \boldsymbol{x} = \boldsymbol{h}^{(i)}\big] = (1-p_k)\mathbb{1}_{\{\boldsymbol{y}_k(t) = \boldsymbol{h}_k^{(i)}\}} + p_k \mathbb{1}_{\{\boldsymbol{y}_k(t) \neq \boldsymbol{h}_k^{(i)}\}} \tag{6}$$

$\mathbb{1}$ is the indicator function, and $\boldsymbol{h}_k^{(i)} \in \{0,1\}$ is the kth entry of $\boldsymbol{h}^{(i)}$. As a result, given the previous posterior $\pi(t-1)$, the action $\mathcal{A}(t)$ and the observation $\boldsymbol{y}_{\mathcal{A}(t)}$, we can exactly compute the updated posterior belief vector $\pi(t)$ using (5).

Before we define the notion of reward, we recall from Section II that our goal is to achieve a balance between the detection accuracy, stopping time, and sensing cost. We can use the posteriors or the MDP states to control the detection accuracy via the stopping rule using a parameter $\pi_{\text{upper}} \in (0,1)$. The parameter π_{upper} represents the threshold on the largest value among the beliefs on the different values of $\boldsymbol{x}$

$$\max_{i=1,2,\ldots,m} \pi_i(T) \geq \pi_{\text{upper}}. \tag{7}$$

Having defined the stopping rule (one of the two components of the algorithm), which controls the detection accuracy, we next focus on the trade-off between the stopping time and sensing cost. This trade-off is determined by the sequential process selection (the other algorithm component), which we derive using the notion of reward and policy.

B. Reward

The reward $r(t)$ at time t is a function of the posterior beliefs $\pi(t)$ and $\pi(t-1)$, and the selected processes $\mathcal{A}(t)$. The reward indicates the intrinsic desirability of choosing the subset of processes as a function of the posterior belief. For a given reward function, the policy $\mu : [0,1]^M \to \mathcal{P}(N)$ is a mapping from the posterior belief vector $\pi(t-1)$ to the processes to be probed $\mathcal{A}(t)$. The policy represents the sequential process selection part of our algorithm, and it is designed to maximize the long-term reward given as follows:

$$R(T) = \sum_{t=1}^{T} \mathbb{E}\{r(t)\}. \tag{8}$$

Here, the expectation is over the uncertainty in the value of $\boldsymbol{x}$ and the observations $\boldsymbol{y}_{\mathcal{A}(t)}(t)$ when $\mathcal{A}(t)$ follows the policy μ.

The reward function balances the trade-off between the stopping time and sensing cost. Here, the sensing cost can be quantified as a function of $\mathcal{A}(t)$ using the term $\sum_{k\in\mathcal{A}(t)} c_k$. However, we also need a term in the reward which forces the policy to build the posterior belief on the true value of $\boldsymbol{x}$ as quickly as possible, and thus minimize the stopping time T. We represent this term using $\xi : [0,1]^M \to \mathbb{R}$ which is a function of the posterior beliefs. We seek ξ that encourages the decision-maker to move away from the non-informative posterior $\pi(t) = 1/M\mathbf{1}$ and move toward the posterior $\pi(t) \in \{\boldsymbol{e}_i\}_{i=1}^{M}$. Here, $\mathbf{1}$ denotes the all-ones vector and $\boldsymbol{e}_i \in \{0,1\}^M$ denotes the ith column of the $M \times M$ identity matrix. Two functions that achieve this goal are given by the following proposition.

Proposition 1: Let $L, H : [0,1]^M \to \mathbb{R}$ be two functions defined, respectively, as

$$L(\pi) = \sum_{i=1}^{M} \pi_i \log \frac{\pi_i}{1-\pi_i} \tag{9}$$

$$H(\pi) = -\sum_{i=1}^{M} \pi_i \log \pi_i. \tag{10}$$

These functions satisfy

$$\begin{aligned} \underset{\substack{\pi\in[0,1]^M \\ \sum_{i=1}^M \pi_i = 1}}{\arg\min}\ L(\pi) &= \underset{\substack{\pi\in[0,1]^M \\ \sum_{i=1}^M \pi_i = 1}}{\arg\max}\ H(\pi) = \frac{1}{M}\mathbf{1} \\ \underset{\substack{\pi\in[0,1]^M \\ \sum_{i=1}^M \pi_i = 1}}{\arg\max}\ L(\pi) &= \underset{\substack{\pi\in[0,1]^M \\ \sum_{i=1}^M \pi_i = 1}}{\arg\min}\ H(\pi) = \{\boldsymbol{e}_i\}_{i=1}^{M} \end{aligned}$$

where $\mathbf{1}$ denotes the all-ones vector and $\boldsymbol{e}_i \in [0,1]^M$ denotes the ith column of the $M \times M$ identity matrix.

Proof: From the log-sum inequality [16], we have

$$\sum_{i=1}^{M} a_i \log\left(\frac{a_i}{b_i}\right) \geq \left(\sum_{i=1}^{M} a_i\right) \log\left(\frac{\sum_{i=1}^{M} a_i}{\sum_{i=1}^{M} b_i}\right)$$

for any set $\{a_i \geq 0, b_i \geq 0\}_{i=1}^{M}$, and equality holds only if $a_i = \alpha b_i$, for some constant $\alpha > 0$. Thus, for any $\pi \in [0,1]^M$

$$\begin{aligned} L(\pi) - L\left(\frac{1}{M}\mathbf{1}\right) &= \sum_{i=1}^{M} \pi_i \log \frac{\pi_i(M-1)}{1-\pi_i} \\ &\geq \left(\sum_{i=1}^{M} \pi_i\right) \log\left(\frac{\sum_{i=1}^{M} \pi_i}{\sum_{i=1}^{M} \frac{1-\pi_i}{M-1}}\right) = 0. \end{aligned}$$

Hence, $L(\pi) \geq L((1/M)\mathbf{1})$ and equality holds only if $\pi_i = 1/M$.

We next look at the maximum of $L(\pi)$ and we see that if $\pi_i \in [0,1)$ for all values of i, $L(\pi) < \infty$. Therefore, $L(\pi)$ attains the maximum value if and only if at least one entry of π is 1. Hence, the desired maxima is achieved at $\{\boldsymbol{e}_i\}_{i=1}^{M}$.

We can compute the maxima and minima of $H(\pi)$ using similar arguments and thus, the proof is complete. ■

The above proposition implies that the functions L and $-H$ are two good choices for ξ. We note that in (9), the term $\log(\pi_i/(1-\pi_i))$ is the LLR of the two hypotheses namely, $\boldsymbol{x} = \boldsymbol{h}^{(i)}$ and $\boldsymbol{x} \neq \boldsymbol{h}^{(i)}$. Consequently, $L(\pi)$ is the Bayesian LLR obtained by applying the logit function on the posterior belief. Further, maximizing the Bayesian LLR increases the posterior belief on the true value of $\boldsymbol{x}$. Also, H is the entropy of the distribution π, and thus, minimizing H reduces the uncertainty in estimation and builds the posterior belief on the true value of $\boldsymbol{x}$.

Having defined the function[1] ξ, we formulate the instantaneous reward of the MDP as a weighted sum of ξ and the sensing cost

$$r(t) = \xi(\pi(t)) - \xi(\pi(t-1)) + \lambda \sum_{k \in \mathcal{A}(t)} c_k \tag{11}$$

where $\lambda > 0$ is the weighing parameter that dictates the balance between the stopping time and the total sensing cost. Thus, from (8) and (11), the long-term expected reward of the MDP up to time t is given by

$$R(t) = \mathbb{E}\left\{ \xi(\pi(t)) - \xi(\pi(0)) - \lambda \sum_{\tau=1}^{t} \sum_{k \in \mathcal{A}(\tau)} c_k \right\}$$

where the expectation is over the distribution of $\boldsymbol{x}$ and the observations $\boldsymbol{y}_{\mathcal{A}(t)}(t)$ given $\mathcal{A}(t)$. The MDP objective is to find a policy or sequence of actions $\{\mathcal{A}(t) \in \mathcal{P}(N)\}_{t=1}^{T}$ that maximizes the long-term average sum of the rewards. Hence, a policy that maximizes the long-term reward improves the accuracy of the estimate [quantified by $\xi(\pi(t))$] as soon as possible while minimizing the overall sensing cost [the last term in $R(t)$]. Further, the agent continues to take observations until it declares an estimate with the desired level of confidence given by π_{upper} (i.e., $t = T$). Therefore, if λ is small, the reward ensures that the agent chooses actions with a significant change $\xi(\pi(t)) - \xi(\pi(0))$, leading to a shorter stopping time. On the other hand, with a large λ, the agent tries to minimize the sensing cost by probing a few processes at every time instant which increases the stopping time. Therefore, λ controls the stopping time and total sensing cost.

Further, the Bayesian LLR L is unbounded unlike the entropy satisfying $H(\pi) \leq \log M$ for any $\pi \in [0,1]^M$. Also,

$$L(\pi) = -H(\pi) - \sum_{i=1}^{M} \pi_i \log(1-\pi_i) \geq -H(\pi). \tag{12}$$

Therefore, for the same value of λ, the Bayesian LLR reward function gives a higher weight to the accuracy than the cost. As a result, the trade-off between the accuracy and sensing cost differs for the two reward functions. We discuss this point in detail in Section VI.

This completes our discussion on the reward function. Using this formulation, we next present the deep learning algorithms to obtain policies that maximize the long-term reward of the MDP. We use two approaches: the deep RL-based approach presented in Section IV and deep AI-based approach presented in Section V.

[1]The algorithmic development is independent of the choice of the reward function (LLR and entropy-based). Therefore, in the remainder of the article, we use $\xi(\cdot)$ in the reward of the MDP which can either be $L(\cdot)$ defined in (9) or $-H(\cdot)$ defined in (10).

IV. Anomaly Detection via Deep RL Algorithms

Our RL algorithms are designed to maximize the expected discounted return $\bar{R}(t)$ defined as

$$\bar{R}(t) = \lim_{T \to \infty} \sum_{j=0}^{T} \gamma^j r(t+j) \tag{13}$$

where $0 < \gamma < 1$ (which is generally close to 1) is the discount factor. This parameter γ weighs the rewards in the distant future relative to those in the immediate future, i.e., a reward received j time steps in the future is worth only γ^{j-1} times what it would be worth if it were received immediately. Hence, this approach encourages the agent to minimize the stopping time.

To maximize $\bar{R}(t)$, the RL algorithms make process selection based on the value functions of the posterior-action pair and the posterior. For a given policy μ, these value functions are

$$Q_\mu(\pi, \mathcal{A}) = \mathbb{E}\left\{ \bar{R}_k \middle| \pi(t-1) = \pi, \mathcal{A}(t) = \mathcal{A} \right\} \tag{14}$$

$$V_\mu(\pi) = \mathbb{E}_{\mathcal{A} \sim \mu(\pi)}\left\{ \bar{R}_k \middle| \pi(t-1) = \pi \right\} \tag{15}$$

where the expectations are evaluated given that the agent follows the policy μ for all future actions. Intuitively, the action-value function (referred to as the Q-function), $Q_\mu(\pi, \mathcal{A})$, in (14) indicates the long-term desirability of choosing a particular action when the posterior belief vector is π. Also, the state-value function (referred to as the value function), $V_\mu(\pi)$, in (15) specifies the expected reward when starting with posterior belief vector π and following the policy μ thereafter. An RL agent makes the action choices by evaluating the optimal value estimates, Q-function or the value function, or both. If we have the optimal values of the functions, then the actions that appear best after a one-step search are the optimal actions [17]. In the following, we present two different RL approaches, the Q-learning and AC algorithms, and describe how they estimate these functions to arrive at the optimal policy.

A. Dueling Deep Q-Learning

The Q-learning approach is a popular RL algorithm where the agent estimates the Q-function and chooses the action $\mathcal{A}(t)$ that maximizes the Q-function given the posterior belief vector $\pi(t-1)$ [18]. Further, in the DQN framework, the unknown Q-function is modeled using a neural network [19], and the dueling DQN framework refers to the implementation of this neural network using a model called the dueling architecture [20].

This architecture consists of a single Q-network and relies on a quantity called the advantage function: $A_\mu(\pi, \mathcal{A}) = Q_\mu(\pi, \mathcal{A}) - V_\mu(\pi)$, which is a measure of how much $Q_\mu(\pi, \mathcal{A})$ deviates from the expected value over all the actions, $V_\mu(\pi)$, and therefore, specifies the relative preference of each action for a given posterior belief vector. The dueling architecture estimates both $V_\mu(\pi)$ and $A_\mu(\pi, \mathcal{A})$ separately and combines them to obtain $Q_\mu(\pi, \mathcal{A})$. The input to the Q-network is the posterior belief vector $\pi \in \mathbb{R}^M$ and the output

Algorithm 1 Dueling Q-Learning Algorithm for Anomaly Detection

Parameters: Prior distribution $\pi(0)$, discount rate $\gamma \in (0, 1)$, and confidence level $\pi_{\text{upper}} \in (0, 1]$

Initialization: Q-network parameter θ_{DQN} arbitrarily

repeat
  Time index $t = 1$
  while $\max_i \pi_i(t-1) < \pi_{\text{upper}}$ **do**
    Choose action $\mathcal{A}(t)$ using the policy derived from $Q(\pi(t), \cdot; \theta_{\text{DQN}})$
    Generate observations $\boldsymbol{y}_{\mathcal{A}(t)}(t)$
    Compute $\pi(t)$ using (5) and $r(t)$ using (11)
    Update θ_{DQN} by minimizing $L_{\text{DQN}}(\theta_{\text{DQN}})$ in (16)
    Increase time index $t = t + 1$
  end while
  Declare $\hat{\boldsymbol{x}} = \boldsymbol{h}_k^{(i^*)}$ where $i^* = \arg\max_i \pi_i(t-1)$
until

is an $(M-1)$-length vector whose ith entry corresponds to $Q_\mu(\pi, \cdot)$ of the ith possible action. The network parameter θ_{DQN} is obtained by optimizing the loss function [20]

$$L_{\text{DQN}}(\theta_{\text{DQN}}) = \mathbb{E}_{\pi,\mathcal{A},\pi'}\Big\{r(t) + \gamma \max_{\mathcal{A}' \in \mathcal{P}(N)} Q(\pi', \mathcal{A}'; \theta^-_{\text{DQN}}) - Q(\pi, \mathcal{A}; \theta_{\text{DQN}})\Big\} \tag{16}$$

where θ^-_{DQN} is the current network parameter estimate obtained in the previous time. We update the Q-function using bootstrapping by basing its update in part on a current estimate $Q(\pi', \mathcal{A}'; \theta^-_{\text{DQN}})$. Also, to ensure that the learned value of the Q-function converges to the optimal Q-function, we use the greedy policy to choose the successor action $\mathcal{A}'$.

Using the learned Q-function, we next describe how to obtain the optimal policy. We derive the policy using a combination of the decaying-epsilon greedy algorithm [20], [21] and the Gibbs softmax method [22]. At every time step, the agent takes action using the Gibbs softmax method with a probability of $1 - \epsilon$ and a random action with a probability of ϵ. Here, ϵ is a parameter that decays with time. Also, the Gibbs softmax method refers to choosing the action $\mathcal{A}(t) \sim \sigma\big(Q(\pi(t), \cdot; \theta_{\text{DQN}})\big) \in [0, 1]^{M-1}$ where $\sigma(\cdot)$ is the softmax function. The approach ensures that the entire action space is explored while exploiting the best action with high probability.

The algorithm chooses actions in the above fashion until the posterior belief distribution satisfies the stopping rule in (7). We present the pseudo-code for our dueling DQN-based detection algorithm in Algorithm 1.

B. Deep Actor–Critic

The deep AC algorithm is another RL algorithm that directly learns the policy. This principle differs from that of the dueling DQN algorithm, which learns the Q-function and derives a policy based on the learned Q-function.

The AC architecture consists of two separate neural networks, actor and critic, with no shared features. The actor learns the policy, which chooses the action based on the posterior probabilities. Thus, its input is $\pi \in \mathbb{R}^M$ and the output is $\mu_{\text{ac}}(\pi, \cdot; \theta_{\text{actor}}) \in [0, 1]^{M-1}$, where θ_{actor} represents the neural network parameters. The policy returned by the actor network is a stochastic policy which chooses an action according to $\mathcal{A} \sim \mu_{\text{ac}}(\pi, \cdot; \theta_{\text{actor}})$. The critic refers to the learned value function, which is an estimate of how good the policy learned by the actor is and hence, essentially provides an evaluation of that policy. The evaluation of the action $\mathcal{A}(t)$ taken corresponding to the posterior $\pi(t-1)$ takes the form of the temporal-difference (TD) error as given by

$$\delta(t) = r(t) + \gamma V_\mu(\pi(t)) - V_\mu(\pi(t-1))$$

for a given policy $\mu(\cdot)$ with

$$V_\mu(\pi) = \mathbb{E}_{\pi',\mathcal{A}\sim\mu(\pi)}\big\{r(t) + \gamma V_\mu(\pi') \big| \pi(t-1) = \pi\big\}.$$

If the TD error is positive, the probability of choosing $\mathcal{A}(t)$ in the future is increased, and vice versa. Therefore, the input to the critic network is the posterior belief vectors $\pi \in [0, 1]^M$ and the output is the corresponding value function $V(\pi; \theta_{\text{critic}}) \in \mathbb{R}$, where θ_{critic} represents the neural network parameters.

We next describe how to learn the two sets of network parameters: θ_{actor} of the actor and θ_{critic} of the critic. Since the goal of the critic network is to fit a model to estimate the optimal value function, its parameter update is equivalent to minimizing the model mismatch between the reward obtained at the current time step and the learned value function. Thus, the critic network updates its parameter θ_{critic} by minimizing the square of the TD error given by

$$\delta(t) = r(t) + \gamma V(\pi(t); \theta_{\text{critic}}) - V\big(\pi(t-1); \theta^-_{\text{critic}}\big) \tag{17}$$

where θ^-_{critic} is the current critic network parameter estimate obtained in the previous time instant. On the other hand, the goal of the actor network is to find a policy that maximizes the value function. Thus, its parameter update is via maximization of the value function. The actor updates its parameter [17] as

$$\theta_{\text{actor}} = \theta^-_{\text{actor}} + \delta(t)\nabla_{\theta_{\text{actor}}}\big[\log \mu_{\text{ac}}(\pi(t-1), \mathcal{A}(t); \theta_{\text{actor}})\big] \tag{18}$$

where θ^-_{actor} is the current actor-network parameter estimate obtained in the previous time instant and $\delta(t)$ given by (17) is obtained from the critic network.

The learned policy is straightforward in the case of the AC framework, as the actor network directly learns the policy. Hence, at every time step, the agent chooses an action based on the output of the actor network and receives the reward for updating both actor and critic networks. The agent stops collecting observations and returns an estimate of $\boldsymbol{x}$ when the confidence level exceeds the desired level, i.e., when (7) holds. The pseudo-code of our algorithm is given in Algorithm 2.

V. Anomaly Detection via Deep Active Inference

The AI framework is an alternate approach to solving the MDP problem described in Section III. It is inspired by a normative theory of brain function based on its perception of the MDP, i.e., the AI agent maintains a generative model that represents its perception [23], [24], [25]. This generative model $\phi(\cdot)$ comprises a joint probability distribution on the posterior, the actions, and the corresponding observations: $\big\{\pi(t-1), \mathcal{A}(t), \boldsymbol{y}_{\mathcal{A}(t)}(t), t > 0\big\}$. The model assigns higher probabilities to the posteriors and actions favorable to the

Algorithm 2 Actor–Critic RL for Anomaly Detection

Parameters: Prior distribution $\pi(0)$, discount rate $\gamma \in (0, 1)$, and confidence level $\pi_{\text{upper}} \in (0, 1]$

Initialization: Actor and critic neural network parameters θ_{actor} and θ_{critic} arbitrarily

repeat
  Time index $t = 1$
  while $\max_i \pi_i(t-1) < \pi_{\text{upper}}$ **do**
    Choose action $\mathcal{A}(t)$ using the policy derived from $\mu_{\text{ac}}(\pi(t-1), \cdot; \theta_{\text{actor}})$
    Generate observations $\boldsymbol{y}_{\mathcal{A}(t)}(t)$
    Compute $\pi(t)$ using (5) and $r(t)$ using (11)
    Update θ_{critic} by minimizing the squared temporal error $\delta^2(t)$ in (17)
    Update θ_{actor} using (18)
    Increase time index $t = t + 1$
  end while
  Declare $\hat{\boldsymbol{x}} = \boldsymbol{h}_k^{(i^*)}$ where $i^* = \arg\max_i \pi_i(t-1)$
until

agent. Given a generative model, the agent inverts the model to find the conditional distribution of the action $\mathcal{A}(t)$ corresponding to the posterior $\pi(t-1)$. However, since directly computing the marginals is difficult, we use the method of approximate Bayesian inference. To this end, it defines a variational distribution $\mu_{\text{AI}}(\pi, \mathcal{A})$ that is controlled by the agent. The distribution $\mu_{\text{AI}}(\cdot)$ is optimized by minimizing the Kullback–Leibler (KL) divergence between the distributions $\mu_{\text{AI}}(\cdot)$ and $\phi(\cdot)$. Therefore, a stochastic policy that chooses actions according to the distribution $\mu_{\text{AI}}(\cdot)$ maximizes the obtained reward. The KL divergence between the variational distribution and the generative model is called the variational free energy. In other words, the goal of the AI agent is to find the stochastic policy $\mu_{\text{AI}}(\cdot)$ which minimizes its expected free energy (EFE).

From (5), we know that the posterior belief vector $\pi(t)$ can be exactly inferred using the knowledge of the action $\mathcal{A}(t)$, observation $\boldsymbol{y}_{\mathcal{A}(t)}$, and posterior belief vector $\pi(t-1)$. This relationship, along with the Markov property, enables us to completely define the generative model using the distribution $\phi(\mathcal{A}(t), \boldsymbol{y}_{\mathcal{A}(t)}(t)|\pi(t-1))$. This distribution is

$$\begin{aligned}&\phi(\boldsymbol{y}_{\mathcal{A}(t)}(t), \mathcal{A}(t)|\pi(t-1))\\ &\quad= \phi(\boldsymbol{y}_{\mathcal{A}(t)}|\mathcal{A}(t), \pi(t-1))\phi(\mathcal{A}(t)|\pi(t-1)).\end{aligned}$$

The generative model is biased toward high rewards, encoded into the generative model as the prior probability of the belief

$$\phi(\boldsymbol{y}_{\mathcal{A}(t)}(t)|\mathcal{A}(t), \pi(t-1)) = \sigma(r(t)) \tag{19}$$

where we recall that $\sigma(\cdot)$ is the softmax function and $r(t)$ denotes the instantaneous reward of the MDP at time t.

We next complete the construction of the generative model by specifying the distribution $\phi(\mathcal{A}(t)|\pi(t))$. Since the agent tries to minimize the total free energy of the expected trajectories into the future, it is encoded into the generative model as

$$\phi(\mathcal{A}(t)|\pi(t)) = \sigma(-G(\mathcal{A}(t), \pi(t-1))) \tag{20}$$

where $G(\cdot)$ is the total free energy of the expected trajectories into the future, and the variational free energy is the KL divergence between the variational distribution $\mu_{\text{AI}}(\cdot)$ and the generative model $\phi(\cdot)$

$$\begin{aligned}F(t) = &\sum_{\mathcal{A}(t)\in\mathcal{P}(N)} \mu_{\text{AI}}(\pi(t-1), \mathcal{A}(t))\\ &\times \log \frac{\mu_{\text{AI}}(\pi(t-1), \mathcal{A}(t))}{\phi(\mathcal{A}(t), \boldsymbol{y}_{\mathcal{A}(t)(t)}|\pi(t-1))}.\end{aligned} \tag{21}$$

Thus, the agent constructs the generative model using the EFE, obtains the optimum policy by minimizing the EFE of all the paths into the future, and chooses an action that minimizes the EFE. In other words, determining the optimal policy reduces to computing and optimizing the EFE.

Next, we present the neural network architecture and learning of the neural network parameters. The deep AI algorithm consists of two neural networks: the policy and EFE. The policy network directly learns the process selection, and therefore, it takes the posterior belief vector $\pi(t-1)$ as the input. Its output is the stochastic selection policy $\mu_{\text{AI}}(\pi(t-1), \cdot; \theta_{\text{policy}}) \in [0, 1]^{M-1}$ which is a probability distribution on $\mathcal{P}(N)$. Here, θ_{policy} denotes the neural network parameters. The EFE network represents EFE's learned value, which estimates how close the learned policy is to the generative model. Thus, the input of the EFE network is the posterior $\pi \in [0, 1]^M$, and the output is the EFE value $G(\pi, \cdot; \theta_{\text{EFE}}) \in \mathbb{R}^{M-1}$, representing the EFE values of each action $\mathcal{A} \in \mathcal{P}(N)$ and the posterior π. Here, θ_{EFE} denotes the parameters of the EFE network.

The EFE can be approximated as follows [26]:

$$G(\mathcal{A}(t), \pi(t-1)) \approx \mathbb{E}\{-r(t) + G(\mathcal{A}(t+1), \pi(t))\}$$

where we use (19). Therefore, we learn the parameters of the EFE network by optimizing the model mismatch between the learned EFE value and the reward obtained

$$\begin{aligned}L_{\text{EFE}}(\theta_{\text{EFE}}) = \mathbb{E}\Big\{&\big(G(\mathcal{A}(t), \pi(t-1); \theta_{\text{EFE}}) + r(t)\\ &- G\big(\mathcal{A}(t+1), \pi(t); \theta_{\text{EFE}}^-\big)\big)^2\Big\}\end{aligned} \tag{22}$$

where θ_{EFE}^- denotes the current estimate of the network parameter obtained in the previous time step and the expectation is over the action distribution $\mathcal{A}(t+1) \sim \mu_{\text{AI}}(\pi(t), \cdot; \theta_{\text{policy}})$. To update the policy network, we minimize the variational free energy defined in (21)

$$\begin{aligned}F(\theta_{\text{policy}}) = &\sum_{\mathcal{A}\in\mathcal{P}(N)} \mu_{\text{AI}}(\pi(t-1), \mathcal{A}; \theta_{\text{policy}})\\ &\times \log \frac{\mu_{\text{AI}}(\pi(t-1), \mathcal{A}; \theta_{\text{policy}})}{\sigma(r(t))\sigma(-G(\pi(t-1), \mathcal{A}; \theta_{\text{EFE}}))}.\end{aligned} \tag{23}$$

Since $r(t)$ is independent of θ_{policy}, the loss function is

$$\begin{aligned}&L_{\text{AI}}(\theta_{\text{policy}})\\ &= -H(\mu_{\text{AI}}(\pi(t-1), \cdot; \theta_{\text{policy}}))\\ &\quad- \sum_{\mathcal{A}\in\mathcal{P}(N)} \mu_{\text{AI}}(\pi(t-1), \mathcal{A}) \log \sigma(G(\pi(t-1), \mathcal{A}; \theta_{\text{EFE}}))\end{aligned} \tag{24}$$

where $H(\cdot)$ is given by (10).

As in the case of the AC algorithm, the policy to be followed by the agent is directly obtained from the (policy) neural network output. The agent follows this policy to choose an action at every time instant, collects the corresponding reward,

and updates the two neural networks using the obtained reward. The algorithm is summarized in Algorithm 3.

Algorithm 3 Active Inference Algorithm for Anomaly Detection

Parameters: Prior distribution $\pi(0)$ and confidence level $\pi_{\text{upper}} \in (0, 1]$

Initialization: Policy and EFE network parameters θ_{policy} and θ_{EFE} arbitrarily

repeat
  Time index $t = 1$
  while $\max_i \pi_i(t-1) < \pi_{\text{upper}}$ **do**
    Choose action $\mathcal{A}(t)$ using the policy derived from $\mu_{\text{AI}}(\pi(t-1), \cdot; \theta_{\text{policy}})$
    Generate observations $\boldsymbol{y}_{\mathcal{A}(t)}(t)$
    Compute $\pi(t)$ using (5) and $r(t)$ using (11)
    Update θ_{EFE} by minimizing $L_{\text{EFE}}(\theta_{\text{EFE}})$ in (22)
    Update θ_{policy} by minimizing the variational free energy $L_{\text{AI}}(\theta_{\text{policy}})$ in (24)
    Increase time index $t = t + 1$
  end while
  Declare $\hat{\boldsymbol{x}} = \boldsymbol{h}_k^{(i^*)}$ where $i^* = \arg\max_i \pi_i(t-1)$
until

A. Comparison With RL Methods

The AI approach has many similarities to RL-based algorithms, such as learning probabilistic models, exploring and exploiting various actions, and efficient planning. In particular, the AI algorithm closely resembles the policy gradient methods (for example, the AC algorithm) since both approaches try to learn the policy directly. We recall that the AC and the AI methods have two separate neural networks. The actor network of the AC algorithm and the policy network of the AI algorithm learn the policy to be followed. In contrast, the other network estimates a function (TD error or EFE) used to evaluate and optimize the policy. However, the two algorithms are derived based on different principles, and the main differences between the RL framework and the AI framework are as follows.

1) Model-Free and Model-Based: The traditional RL uses a model-free approach where the algorithm aims at reward maximization based on the Q function or the value function, or both. The algorithm does not try to explicitly learn the probabilistic model that governs the state transition or the generation of the observations of the MDP. However, the AI model relies on a hierarchical generative model, which is based on variational free energy. It explicitly learns the model consisting of states, actions, and observations of the MDP. To be specific, in the most general setting, the deep AI algorithm consists of four neural networks: policy network; EFE network; observation network to learn the distribution of the observations given the state and action; and state transition network to learn the distribution of the next state given the previous state, action, and observation [26]. However, in our case, (19) and (5) define the distributions learned by the observation network and the state transition network. Hence, the AI algorithm comprises only two neural networks. In other words, the AI algorithm naturally allows the algorithm to incorporate any knowledge of the environment's statistics into the model.

2) Policy Optimization: The AC algorithm uses the value functions directly to learn the policy. In contrast, the AI algorithm relies on the generative probability distribution derived from softmax over the variational free energy as defined in (20). Therefore, the AC algorithm maximizes the expected reward function in the future, whereas the AI algorithm reduces the surprise in the future by learning the probabilistic model. Moreover, the objective function of the AC algorithm depends only on the samples generated using the actions that the agent took within the episode as opposed to the AI algorithm, which averages the objective function over all possible actions in the next step [see the summation in (24)]. Since the AI algorithm computes the expected value, it may lead to reduced variance and better performance.

B. Comparison With Chernoff Test

The Chernoff test, a standard algorithm for active hypothesis testing [5], sequentially chooses actions that build the posterior belief on the true value of $\boldsymbol{x}$ as quickly as possible. However, it does not take the sensing cost into account. It follows the stochastic policy μ_{Chernoff}

$$\mu_{\text{Chernoff}}(\pi(t-1), \cdot) = \arg\max_{\substack{\boldsymbol{q}\in[0,1]^{M-1} \\ \sum_i \boldsymbol{q}_i = 1}} \min_{\hat{\boldsymbol{x}}\in\{0,1\}^M \atop \hat{\boldsymbol{x}}\neq\bar{\boldsymbol{x}}(t-1)} \boldsymbol{q}^{\top}\boldsymbol{d}(\bar{\boldsymbol{x}}(t-1), \hat{\boldsymbol{x}}) \tag{25}$$

where we define the ith entry of $\boldsymbol{d}(\cdot) \in \mathbb{R}^{M-1}$ as

$$\boldsymbol{d}_i(\bar{\boldsymbol{x}}(t), \hat{\boldsymbol{x}}) \triangleq \text{KL}\big(p\big(y_{\mathcal{A}_i}(t)|\boldsymbol{x} = \bar{\boldsymbol{x}}(t)\big) \| p\big(y_{\mathcal{A}_i}(t)|\boldsymbol{x} = \hat{\boldsymbol{x}}\big)\big)$$
$$\bar{\boldsymbol{x}}(t) = \boldsymbol{h}^{(\tilde{i}^*)}; \quad \tilde{i}^* = \arg\max_i \pi_i(t).$$

Here, KL denotes the KL divergence between the distributions, and $\mathcal{A}_i$ denotes that ith element of set $\mathcal{P}(N)$. However, for any $\hat{\boldsymbol{x}}$, the KL divergence term is maximized when all the processes are selected. Thus, we have $\boldsymbol{d}_i(\bar{\boldsymbol{x}}(t), \hat{\boldsymbol{x}}) \leq \boldsymbol{d}_1(\bar{\boldsymbol{x}}(t), \hat{\boldsymbol{x}})$ with $\mathcal{A}_1 = \{1, 2, \ldots, N\}$ denoting the action of selecting all processes. Therefore, we arrive at

$$\max_{\substack{\boldsymbol{q}\in[0,1]^{M-1} \\ \sum_i \boldsymbol{q}_i = 1}} \min_{\hat{\boldsymbol{x}}\in\{0,1\}^M \atop \hat{\boldsymbol{x}}\neq\bar{\boldsymbol{x}}(t-1)} \boldsymbol{q}^{\top}\boldsymbol{d}(\bar{\boldsymbol{x}}(t-1), \hat{\boldsymbol{x}}) \leq \min_{\hat{\boldsymbol{x}}\in\{0,1\}^M \atop \hat{\boldsymbol{x}}\neq\bar{\boldsymbol{x}}(t-1)} \boldsymbol{d}_1(\bar{\boldsymbol{x}}(t), \hat{\boldsymbol{x}})$$

and equality holds when $\boldsymbol{q} = \begin{bmatrix}1 & 0 & \cdots & 0\end{bmatrix} \in [0, 1]^{M-1}$. Therefore, the Chernoff test always chooses the action $\mathcal{A}_1$ with probability one. This policy is expected as the Chernoff test does not optimize the sensing cost, and thus, it achieves a small stopping time while incurring a high sensing cost. On the contrary, our formulation balances the trade-off between the stopping time and sensing cost via λ and exploits the statistical dependence in $\boldsymbol{x}$ modeled using $\pi(0)$. We corroborate this point using results in Section VI.

VI. Numerical Results

In this section, we present numerical results comparing the performances of deep RL and deep AI algorithms. We choose the number of processes as $N = 3$ and, thus, $M = 2^N = 8$. The prior probability of a process being normal is taken as

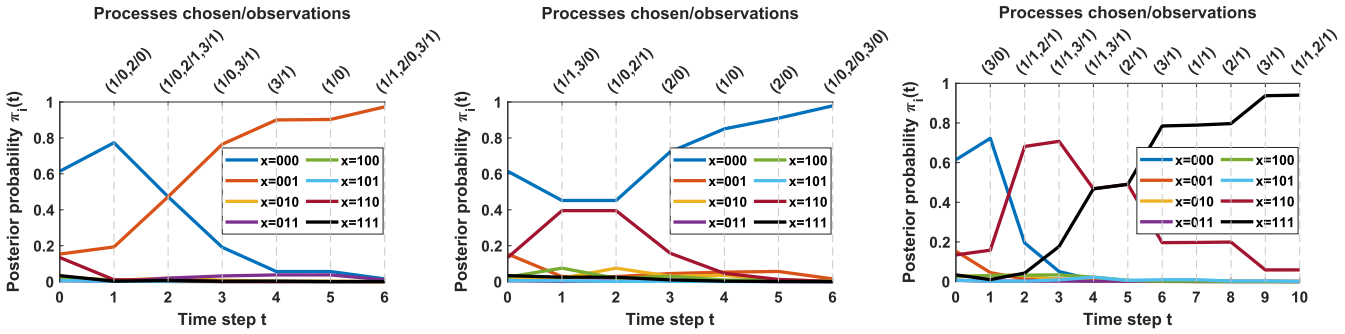


Fig. 2. Single realization of the variation of the belief vector $\pi(t)$, sensor selection $\mathcal{A}(t)$, and the corresponding observations $\boldsymbol{y}_{\mathcal{A}_t}(t)$ over time t. We choose $\pi_{\text{upper}} = 0.94$, $\rho = 0.8$, and $\lambda = 0.2$. The curves represent the evolution of the posterior probabilities of different hypotheses. The selected processes (sensors) and the corresponding observations at different times are depicted at the top of the figure. The figure shows how the posterior probability corresponding to the true hypothesis grows evolves over time and exceeds the desired confidence level π_{upper}. (a) Actor–critic's policy when the process state is [0 0 1]. (b) Active inference's policy when the process state is [0 0 0]. (c) Deep Q-learning's policy when the process state is [1 1 1].

$q = 0.8$. Here, the first and second processes are assumed to be statistically dependent, and the third is independent of the other two. The correlation between the dependent processes is captured by the parameter $\rho \in [0, 1]$

$$\mathbb{P}\{\boldsymbol{x} = [0 \quad 0]\} = q^2 + \rho q(1-q)$$
$$\mathbb{P}\{\boldsymbol{x} = [0 \quad 1]\} = \mathbb{P}\{\boldsymbol{x} = [1 \quad 0]\} = q(1-q)(1-\rho).$$

Also, we assume that the maximum number of time slots for each episode (trial or run) is $T_{\max} = 5000$.

We implement all the neural networks (the Q-network of DQN, the actor and critic networks, and the policy and the bootstrapped EFE networks of AI) with three layers and the ReLU activation function between each consecutive layer. To update the network parameters, we apply the Adam Optimizer. Also, we set $\gamma = 0.9$ for the RL algorithms, and ϵ values linearly decrease from 0.4 to 0.05.

We train the neural networks over multiple episodes (realizations) where, for each episode, we choose the process states from the prior distribution mentioned above, and the number of time slots for each episode is fixed as 50. The AC and AI algorithms converge after 1000 episodes, whereas the DQN algorithm requires 2000 episodes to achieve a stable policy. So, the DQN algorithm requires more extended training than the other two algorithms.

After the training phase, we test the algorithms. We start with three illustrations in Fig. 2. They show the realizations of the variation of the belief vector $\pi(t)$, sensor selection $\mathcal{A}(t)$, and the corresponding observations $\boldsymbol{y}_{\mathcal{A}(t)}(t)$ over time k until the stopping time. Fig. 2(a) shows the sensor selection of the AC algorithm when the true hypothesis (process state) is [0 0 1]. Here, the posterior probability corresponding to the wrong process state [0 0 0] was high initially due to the prior distribution. Since the true process state [0 0 1] and the state [0 0 0] differ only in the state of the third process, the posterior probability corresponding to the true process state [0 0 1] is not high until the third process is observed at $t = 2$. Note that at $t = 2$, the selected processes and the corresponding observations are described by $(1/0, 2/1, 3/1)$, indicating that all three processes have been chosen and the noisy observations are [0 1 1]. As the probability of the true process state [0 0 1] increases (at time $t = 2$), the algorithm observes the third process more often. Finally, at time $t = 6$, the posterior probability of the true process state [0 0 1] exceeds $\pi_{\text{upper}} = 0.94$ and the algorithm stops. We can make similar observations from Fig. 2(b) where the true process state is [0 0 0]. Due to the error in observations from the first process at $t = 1$ and the second process at $t = 2$, the posterior probability of the state [1 1 0] increases initially. Then, the algorithm observes these two processes more often. This policy allows the algorithm to observe that the probability of the state [1 1 0] decreases, and the probability of the true process state [0 0 0] exceeds π_{upper} at $t = 6$. Likewise, in Fig. 2(c), the actual hypothesis is [1; 1; 1]. Nevertheless, up to $t = 6$, the true hypothesis remains uncertain between [1; 1; 0] and [1; 1; 1]; so it is validated by probing the third process at both $t = 6$ and $t = 9$.

Next, we show the performance of the algorithms. Like the training phase, for each episode of the testing phase, we choose the process states from the prior distribution. The three performance metrics we use for comparison are detection accuracy, stopping time, and total cost, as defined in Section II. If the estimated hypothesis is the same as the true hypothesis, the (instantaneous) detection accuracy is one, and otherwise, it is zero. Also, stopping time is the shortest time at which the stopping criteria in (7) is met. The average detection accuracy, stopping time, and total cost obtained using the 10^4 episodes are shown in Figs. 3–7. Like the training phase, during testing for each episode, we choose the process states from the prior distribution mentioned above. In Figs. 4–6, we also show bar plots where the heights are proportional to the fraction of times each process is chosen. In the figures, we compare the three algorithms (label names in brackets), DQN, AC, and AI algorithms considering both Bayesian LLR-based (`LLR`) and entropy-based (`Entropy`) reward functions. Our observations from the numerical results are presented next.

1) Confidence Level π_{upper}: The variations in the performance of different algorithms with π_{upper} are shown in the first row of Fig. 3, and Figs. 4–6. All three performance metrics increase with π_{upper} in all cases. This observation is intuitive as a higher value of π_{upper} implies higher accuracy and requires the algorithms to collect more observations before they decide

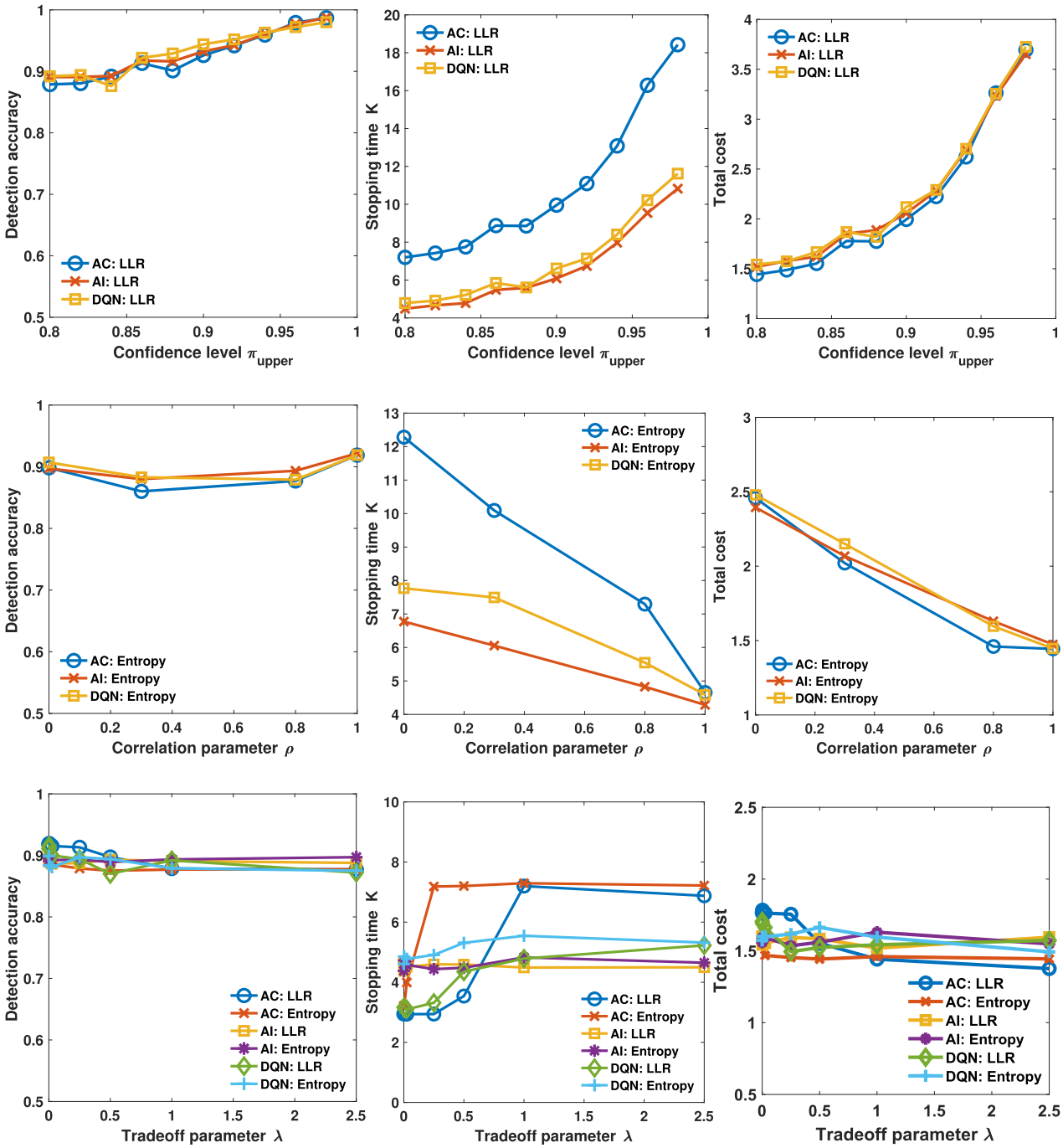


Fig. 3. Performance of the AC, AI, and DQN algorithms for two different reward functions. Unless otherwise mentioned in the plot, we choose $\pi_{\text{upper}} = 0.8$, $\rho = 0.8$, and $\lambda = 1$. The main findings from the figure are as follows: 1) detection accuracy depends only on the confidence level π_{upper}; 2) stopping time and total cost reduce with the correlation co-efficient ρ; and 3) total cost is insensitive to the trade-off parameter λ while the stopping time increases with λ.

on anomalous processes. Also, the accuracy levels achieved by all the algorithms are comparable in all the settings because the common π_{upper} sets the desired confidence level of detection.

2) Correlation Parameter ρ: The second row of Fig. 3 illustrates the performances with varying ρ. The accuracy is insensitive to ρ as it is decided by the confidence level π_{upper}. On the other hand, the stopping time and total cost decrease with ρ. This decrease is expected because when the correlation increases, an observation corresponding to one of the dependent processes gives more information about the other. Consequently, the algorithms require fewer observations and a shorter stopping time to reach the same confidence level.

3) Tradeoff Parameter λ: The last row of Fig. 3 depicts the changes in the algorithm performances with λ. As in the case of ρ, the accuracy and total cost do not vary significantly with λ for a fixed value of π_{upper} and ρ. This behavior is because when ρ is fixed, we need the same number of observations to achieve the same confidence level. However, as λ increases, each observation becomes costlier, and the

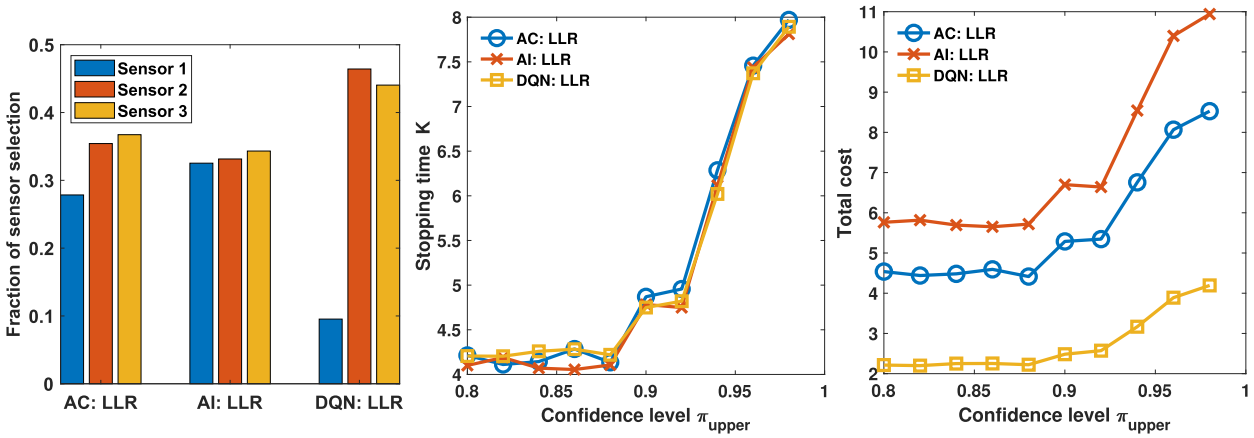


Fig. 4. Performance of the AC, AI, and DQN algorithms when the sensing costs differ: $c_1 = 2$ and $c_2 = c_3 = 0.2$. We choose $\rho = \lambda = 1$, $p_i = 0.2$ for $i = 1, 2, 3$, and for the bar plot, we set $\pi_{\text{upper}} = 0.94$. The figure reveals that, in comparison to other methods, DQN demonstrates superior adaptability to fluctuating costs.

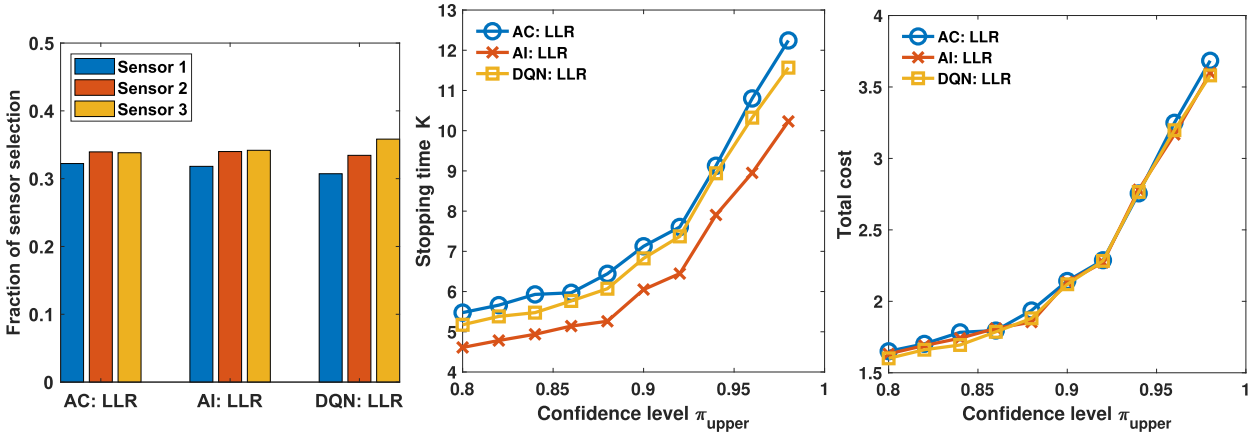


Fig. 5. Performance of the AC, AI, and DQN algorithms when the flipping probabilities differ: $p_1 = 0.45$ and $p_2 = p_3 = 0.2$. We choose $\rho = \lambda = 1$, $c_i = 0.2$ for $i = 1, 2, 3$, and for the bar plot, we set $\pi_{\text{upper}} = 0.94$. It indicates that AI exhibits superior adaptation to varying flipping probability compared to the other methods.

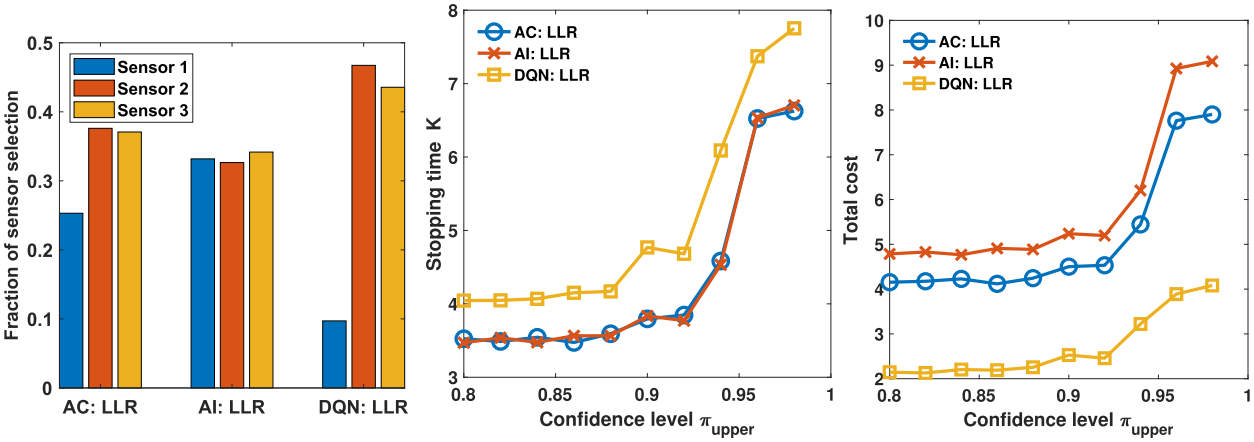


Fig. 6. Performance of the AC, AI, and DQN algorithms when both sensing costs are different: $c_1 = 2$ and $c_2 = c_3 = 0.2$; and $p_1 = 0.02$ and $p_2 = p_3 = 0.2$. We choose $\rho = \lambda = 1$, and we set $\pi_{\text{upper}} = 0.94$ for the bar plot. It demonstrates that in a heterogeneous setting, DQN emerges as the most cost-effective solution, albeit with the drawback of incurring the longest stopping time.

stopping time increases. We notice that the stopping time of the AC algorithm is more sensitive to λ compared to the DQN and AI algorithms. One reason for this could be that the temporal error, which is a function of only the posterior, is more sensitive to the parameter λ than the Q-function learned by the DQN algorithm and EFE learned by the AI algorithm, which are both functions of the posterior belief and action.

4) Reward Functions: From Fig. 3, we infer that all the algorithms provide similar performance levels with both choices of the reward function. To compare the two rewards, we examine the curves associated with AC, which is particularly responsive to cost considerations. Referring to (12), we observe that for identical λ values, the Bayesian LLR reward function assigns greater weight to accuracy than cost. Consequently, the trade-off between accuracy and sensing cost

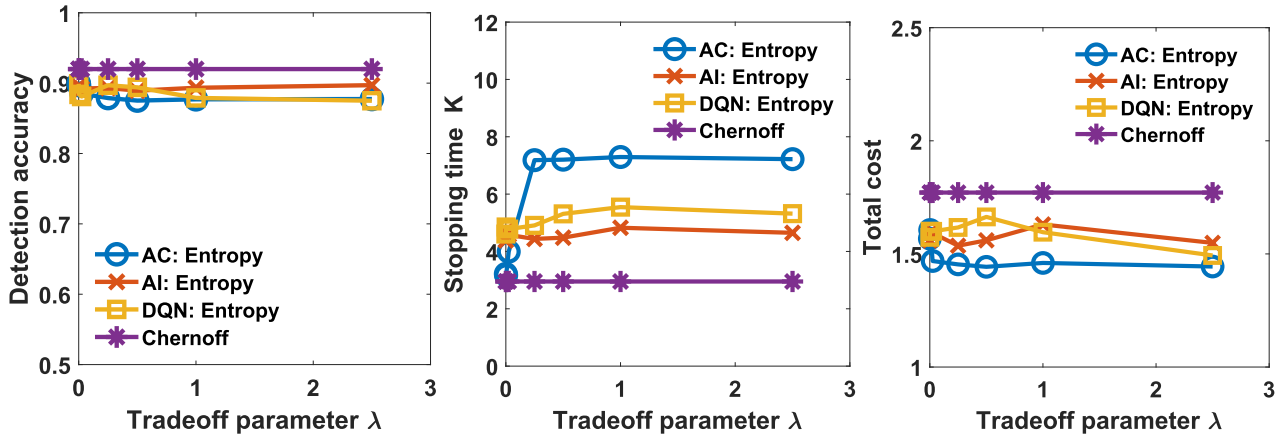


Fig. 7. Comparison of our algorithms with the Chernoff test when $\pi_{\text{upper}} = 0.8$, $\rho = 0.8$, and $c_i = p_i = 0.2$ for $i = 1, 2, 3$. The figure suggests that the Chernoff test fails to adjust the sensing cost, whereas our algorithms can adapt, resulting in reduced total cost.

differs for the two reward functions, as evident in the stopping time (as a function of λ in the bottom middle subfigure) in Fig. 3. For a given λ value, the LLR reward function results in higher cost and lower stopping time due to its reduced sensitivity to λ. For example, the sudden change in the stopping time of the AC algorithm with λ occurs at $\lambda = 0.05$ for the entropy-based function, whereas it occurs at $\lambda = 0.2$ for the Bayesian LLR-based reward. As a result, the performance with the entropy-based function for a particular value of λ is similar to that with the Bayesian LLR reward for a larger value of λ. Moreover, as λ approaches infinity, the effective reward prioritizes sensing cost optimization over accuracy. However, minimizing sensing cost entails selecting a single sensor (minimum) at each time instant. The bottom rightmost subfigure in Fig. 3 indicates that for large λ values, the total cost is approximately 1.5, implying around $1.5/0.2 = 7.5$ observations (since the cost per observation is 0.2), nearly equal to the stopping time. Consequently, the algorithm consistently opts for a single process at each time instant, aligning with expectations.

5) Sensing Cost c_i and Flipping Probability p_i: We analyze the dependence of the algorithms' performance on the sensing cost and flipping probability under three settings: 1) nonuniform sensing costs and uniform flipping probabilities (see Fig. 4); 2) uniform sensing costs and nonuniform flipping probabilities (see Fig. 5); and 3) nonuniform sensing costs and flipping probabilities (see Fig. 6) across the processes. From Fig. 4, the DQN algorithm is more sensitive to different cost values c_i. In all settings considered in Fig. 4, the DQN agent chooses the first process less often, leading to the lowest total cost and best performance. The AC algorithm also adapts to the varying cost, while the AI algorithm is relatively less insensitive to the different costs. Similarly, when we increase the flipping probability of the first process in Fig. 5 (with uniform sensing costs), we see that all algorithms adapt their policies. However, the policy offered by the AI algorithm has shorter stopping times than the other algorithms for comparable values of the total cost. The differences in the policies of the three algorithms are more evident in Fig. 6 when we vary both sensing cost and flipping probabilities. In this setting, the DQN algorithm chooses the first process less often despite its smaller flipping probability. As in the case of Fig. 5, AI is more sensitive to the flipping probability than the cost, and as a result, it gives the shortest stopping times at the price of a higher total cost. The performance of the AC algorithm is between those of the other two algorithms. The AC algorithm provides stopping times comparable to those of the AI algorithm while incurring a smaller total sensing cost.

6) Competing Algorithms: We first note that all the algorithms have similar detection accuracy due to the common stopping criteria in (7), i.e., they stop only when the detection accuracy of the algorithm always exceeds π_{upper}. So, the choice of the best learning algorithm depends on the stopping time and total cost. We first look at the algorithm performances for the uniform cost and filliping probability case from Figs. 3 and 7. For small values of λ, the AC algorithm offers the best stopping time but has a slightly higher cost than the other algorithms. As λ increases, its stopping time also increases, and the AI algorithm provides the best stopping time for a comparable total cost. Also, the AI algorithm offers slightly better performance than DQN. However, our experiments show that the Q-learning algorithm requires more episodes in the training phase than the other algorithms to achieve a stable policy. The memory replay in the Q-learning algorithm also makes its training phase further longer than the other algorithms. Therefore, the AC algorithm is more suitable for sensing cost-critical applications, and for time-sensitive applications, we recommend the AI algorithm over Q-learning.

Next, we look at the nonuniform setting in Figs. 4–6. We notice that the DQN algorithm is more sensitive to the nonuniform sensing cost, whereas the AI algorithm is more sensitive to the nonuniform flipping probability. So, in the nonuniform setting, we prefer Q-learning for cost-critical applications and AI for stopping time-sensitive applications. These observations further justify our joint analysis of different learning-based methods.

7) Comparison With Chernoff Test: Fig. 7 compares our algorithms with the classical Chernoff test. The Chernoff test does not account for the correlation and uses the same strategy all the time. For example, when the correlation parameter

$\rho = 0.8$, the sensing cost incurred by the Chernoff test is 1.85, implying that the average number of observations is $1.85/0.2 = 9.25$ (since the cost per observation is 0.2) and resulting in a stopping time of approximately 3.08. On the other hand, the AC algorithm achieves a total cost of 1.48, reducing the number of observations to 7.4, which is 20% less than that of the Chernoff test. In other words, the Chernoff test fails to adjust to the correlation parameter and the tradeoff parameter λ, whereas our algorithms can capitalize on the correlation and adjust to the tradeoff parameter, balancing the stopping time and total cost. Notably, as the Chernoff test does not account for λ, its performance remains constant across all λ values, while our algorithms consistently exhibit lower sensing costs compared to the Chernoff test.

VII. Conclusion

This article addressed the challenge of anomaly detection, aiming to identify anomalies within a given set of processes. The problem was formulated as an MDP, with the objective of achieving detection accuracy surpassing a predefined threshold while minimizing both delay and total sensing cost. Two distinct objective functions, grounded in Bayesian LLR and entropy, were developed. The study introduced two deep reinforcement learning-based algorithms, namely dueling DQN and AC, along with a deep AI algorithm. Simulation results were presented to compare the efficacy of these algorithms, revealing that our methods outperformed the traditional Chernoff test. Both reward functions exhibited commendable performance, with LLR reward function providing a higher weight to the accuracy than the cost. Different algorithmic approaches had their advantages from different perspectives. Notably, the dueling DQN algorithm necessitated an extended training phase, while the remaining two approaches displayed similar training overheads. Furthermore, the AC and dueling DQN algorithms demonstrated favorable stopping times at a slightly increased sensing cost. In contrast, the AI algorithm exhibited greater resilience to the trade-off parameter and adapted more effectively to the correlation parameter and flipping probabilities. Consequently, the reinforcement learning algorithm proved more suitable for applications where sensing cost is critical, while the AI algorithm is recommended for time-sensitive applications. As a prospective avenue of research, extending our algorithm to monitor behavioral changes in processes over an extended time frame represents an intriguing direction for future exploration.

References

[1] W.-Y. Chung and S.-J. Oh, "Remote monitoring system with wireless sensors module for room environment," *Sens. Actuators B, Chem.*, vol. 113, no. 1, pp. 64–70, Jan. 2006.

[2] A. Bujnowski, J. Ruminski, A. Palinski, and J. Wtrorek, "Enhanced remote control providing medical functionalities," in *Proc. Int. Conf. Pervasive Comput. Technol. Healthcare Workshops*, May 2013, pp. 290–293.

[3] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep actor-critic reinforcement learning for anomaly detection," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.

[4] G. Joseph, M. C. Gursoy, and P. K. Varshney, "Anomaly detection under controlled sensing using actor-critic reinforcement learning," in *Proc. IEEE Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, May 2020, pp. 1–5.

[5] H. Chernoff, "Sequential design of experiments," *Ann. Math. Statist.*, vol. 30, no. 3, pp. 755–770, Sep. 1959.

[6] M. Naghshvar and T. Javidi, "Extrinsic Jensen–Shannon divergence with application in active hypothesis testing," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2012, pp. 2191–2195.

[7] M. Naghshvar and T. Javidi, "Active sequential hypothesis testing," *Ann. Statist.*, vol. 41, no. 6, pp. 2703–2738, Dec. 2013.

[8] M. Franceschetti, S. Marano, and V. Matta, "Chernoff test for strong-or-weak radar models," *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 289–302, Jan. 2017.

[9] B. Huang, K. Cohen, and Q. Zhao, "Active anomaly detection in heterogeneous processes," *IEEE Trans. Inf. Theory*, vol. 65, no. 4, pp. 2284–2301, Apr. 2019.

[10] D. Kartik, E. Sabir, U. Mitra, and P. Natarajan, "Policy design for active sequential hypothesis testing using deep learning," in *Proc. 56th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2018, pp. 741–748.

[11] G. Joseph, C. Zhong, M. C. Gursoy, S. Velipasalar, and P. K. Varshney, "Anomaly detection via controlled sensing and deep active inference," in *Proc. (GLOBECOM) IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6.

[12] T. Yaacoub, G. V. Moustakides, and Y. Mei, "Optimal stopping for interval estimation in Bernoulli trials," *IEEE Trans. Inf. Theory*, vol. 65, no. 5, pp. 3022–3033, May 2019.

[13] P. Grambsch, "Sequential sampling based on the observed Fisher information to guarantee the accuracy of the maximum likelihood estimator," *Ann. Statist.*, vol. 11, no. 1, pp. 68–77, Mar. 1983.

[14] P. J. Bickel and J. A. Yahav, "Asymptotically pointwise optimal procedures in sequential analysis," in *Proc. Berk. Symp. Math. Statist. Probab*, vol. 1, 1967, pp. 401–413.

[15] G. Joseph, M. C. Gursoy, and P. K. Varshney, "Temporal detection of anomalies via actor-critic based controlled sensing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 1–6.

[16] T. M. Cover, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 1999.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[18] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Psychol. Dept., King's College, Cambridge, U.K., 1989.

[19] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 7540.

[20] Z. Wang, N. D. Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 48, Jun. 2016, pp. 1995–2003.

[21] A. Ostovar, O. Ringdahl, and T. Hellström, "Adaptive image thresholding of yellow peppers for a harvesting robot," *Robotics*, vol. 7, no. 1, p. 11, Feb. 2018.

[22] W. Kong, W. Krichene, N. Mayoraz, S. Rendle, and L. Zhang, "Rankmax: An adaptive projection alternative to the softmax function," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 633–643.

[23] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, and G. Pezzulo, "Active inference: A process theory," *Neural Comput.*, vol. 29, no. 1, pp. 1–49, Jan. 2017.

[24] K. J. Friston, M. Lin, C. D. Frith, G. Pezzulo, J. A. Hobson, and S. Ondobaka, "Active inference, curiosity and insight," *Neural Comput.*, vol. 29, no. 10, pp. 2633–2683, Oct. 2017.

[25] K. Friston, F. Rigoli, D. Ognibene, C. Mathys, T. Fitzgerald, and G. Pezzulo, "Active inference and epistemic value," *Cognit. Neurosci.*, vol. 6, no. 4, pp. 187–214, Oct. 2015.

[26] B. Millidge, "Deep active inference as variational policy gradients," *J. Math. Psychol.*, vol. 96, Jun. 2020, Art. no. 102348.