Can you trust your Agent? The Effect of Out-of-Distribution Detection on the Safety of Reinforcement Learning Systems

Tom Haider
Fraunhofer IKS
Munich, Germany

Karsten Roscher
Fraunhofer IKS
Munich, Germany

Benjamin Herd
Fraunhofer IKS
Munich, Germany

Felippe Schmoeller Roza
Fraunhofer IKS
Munich, Germany

Simon Burton
Fraunhofer IKS
Munich, Germany

ABSTRACT

Deep Reinforcement Learning (RL) has the potential to revolutionize the automation of complex sequential decision-making problems. Although it has been successfully applied to a wide range of tasks, deployment to real-world settings remains challenging and is often limited. One of the main reasons for this is the lack of safety guarantees for conventional RL algorithms, especially in situations that substantially differ from the learning environment. In such situations, state-of-the-art systems will fail silently, producing action sequences without signalizing any uncertainty regarding the current input. Recent works have suggested Out-of-Distribution (OOD) detection as an additional reliability measure when deploying RL in the real world. How these mechanisms benefit the safety of the entire system, however, is not yet fully understood. In this work, we study how OOD detection contributes to the safety of RL systems by describing the challenges involved with detecting *unknown* situations. We derive several definitions for unknown events and explore potential avenues for a successful safety argumentation, building on recent work for safety assurance of Machine Learning components. In a series of experiments, we compare different OOD detectors and show how difficult it is to distinguish harmless from potentially unsafe OOD events in practice, and how standard evaluation schemes can lead to deceptive conclusions, depending on which *definition of unknown* is applied.

KEYWORDS

Reinforcement Learning; OOD Detection; Anomaly Detection; AI Safety; Sequential Decision Making

ACM Reference Format:
Tom Haider, Karsten Roscher, Benjamin Herd, Felippe Schmoeller Roza, and Simon Burton. 2024. Can you trust your Agent? The Effect of Out-of-Distribution Detection on the Safety of Reinforcement Learning Systems . In *Proceedings of ACM SAC Conference (SAC'24).* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3605098.3635931

ACM ISBN 979-8-4007-0243-3/24/04…$15.00
https://doi.org/10.1145/3605098.3635931

1 MOTIVATION

Deep Reinforcement Learning (RL), the combination of *Deep Learning* and *Reinforcement Learning*, is an approach to end-to-end learning of sequential decision-making agents via trial and error, where key components (e.g., value-function, policy, world-model) are modeled via deep neural networks. RL has been successfully applied to a series of high dimensional problems, such as dexterous manipulation of robots [31], autonomous navigation of stratospheric balloons [5] and closed-loop blood glucose control [19]. However, plain RL algorithms are built in a way that they return an output, no matter which inputs they receive. This results in a plethora of safety concerns that need to be addressed when deploying RL in real-world applications. In real-world applications, we have to assume that the agent may encounter entirely novel situations. Such samples can cause learning-based components to produce completely irrational outputs. Algorithms purely intended for decision-making will not "notice" unknown situations but over-confidently predict control outputs nonetheless. This was shown to be especially problematic for Deep Neural Networks (DNN)[34]. In safety-critical applications, this necessitates the use of a safety-monitor, that prevents unsafe behavior caused by unknown inputs.

But what can be considered as *unknown* input? Certainly, the safety-monitor should not intervene if the environment changes only slightly, and the policy can still accomplish the task as before. Also, whether unsafe behavior can be attributed to unknown inputs and not to other insufficiencies is not trivial. And, what exactly can be considered as unsafe behavior? Existing work has tackled these questions in a relatively isolated manner either from a pure Machine Learning (ML) or Safety perspective. For example, [22, 36] study Out-of-Distribution (OOD)-detection for RL agents. While these methods have been shown to be capable of detecting perturbations in the environment of RL agents, their impact on overall safety has not yet been studied. Vice versa, works such as [32] studied the Safety of Decision-Making in Autonomous Systems, but did not include situations outside the Operational Domain Model (ODM).

In this work, we approach the problem of safety for RL agents in light of unknown situations. Our goal is neither to provide a novel methodology for Out-of-Distribution (OOD)-detection in RL nor a formal method for verifying safety. Instead, this paper aims to combine two different perspectives on the problem (safety and ML), that have so far only been considered in isolation. To the best of our knowledge, there does not exist any work that examines the safety aspects of OOD-detection mechanisms in the context of RL. The main goal of this paper is, therefore, to provide a bridge between the safety and the ML world.

We do this by outlining and structuring existing work from the safety assurance and RL literature related to the problem (Section 3). We then analyze OOD-Detection from a safety perspective and place it within an established safety context (Section 4). Since there is currently no clear definition of OOD events in RL, we explore several possible approaches for this (Section 5). In a series of experiments we then illustrate the relationship between the *definition of unknown*, *choice of detector* and *choice of safety metric* and their impact on safety. In particular, we show how difficult it is to distinguish harmless and potentially harmful OOD events in practice, and how standard evaluation schemes can lead to deceptive conclusions, depending on which *definition of unknown* [1] is applied (Section 6).

2 PRELIMINARIES AND ASSUMPTIONS

We consider RL agents as decision-making systems that sequentially interact with their environment by taking actions. This is formalized as a discrete-time Markov Decision Process (MDP) [6], captured via the tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, r, f, \mu_0)$. $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, and $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ the reward function. $f : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the transition function which describes the system dynamics and $\mu_0 : \mathcal{S} \mapsto [0, 1]$ is the starting state distribution. At each time-step the agent observes the current state $s_t \in \mathcal{S}$, takes an action $a_t \in \mathcal{A}$ and transitions to the next state $s_{t+1} \in \mathcal{S}$ according to the dynamics function f. In non-deterministic settings, f underlies a random process with a probability distribution $P(s_{t+1} \mid s_t, a_t)$. The RL objective is to maximize the expected return

$$\underset{\pi}{\text{maximize}}\ J_{RL}(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k)\right] \tag{1}$$

along the trajectories $\tau = (s_0, a_0, s_1, a_1, \dots)$ produced under policy $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ with discount factor $\gamma \in (0, 1)$. The shortcut $\tau \sim \pi$ describes trajectories generated under policy π.

Throughout this work we make the following assumptions:

- **A1: Markov Condition** Unless otherwise specified, we assume that each state includes information about all aspects of the past agent–environment interaction that have an influence on the future.
- **A2: Full observability** Unless otherwise specified, we assume that the agent directly observes the underlying state of the environment
- **A3: Determinism** We only consider deterministic MDPs.

A1 and A2 are often justifiable, even if not all environment parameters are included in an observation - if all non-observable environment parameters are constant, they can be inferred from interaction. However, if the environment is subject to some perturbation, this is not the case anymore, and the environment becomes partially observable.

We assume the following training paradigm. An RL agent is initially trained in a *training environment*, where the goal is to minimize unsafe interactions while maximizing reward. We do not address the problem of safe exploration, so the agent can explore "unsafely" during this stage. This may be achieved through end-to-end learning, but also with the help of additional safety mechanisms. Once an acceptable level of performance (both in terms of rewards and safety) is achieved, the policy parameters are frozen and the agent is deployed within the *deployment environment*.

Although the training environment should resemble the deployment environment as closely as possible, conditions can still occur in the deployment environment that were not experienced during training, i.e. unknown events. These unknown inputs can cause the agent to exhibit unsafe behaviour. The subject of this work is to study methods that aim to detect and intercept such unknown events, s.t. safety violations are mitigated. It is important to note that, if a situation is unknown, this does not follow that it will lead to a harmful decision. Without further assumptions, this causality cannot be established.

3 RELATED WORK

At an abstract level, safety can be defined as the absence of unreasonable risk of harm. Risk, in turn, represents the product of the probability of a harmful event and its severity. In this paper, we focus on risk caused by insufficiencies in the function (RL agent), i.e. even in the absence of hardware defects or systematic design errors, the function is insufficient to fulfill its safety requirements.

There is a vast body of work from the field of safety engineering, that addresses safety assurance for ML components via system architectures, processes and design principles. In the field of RL, safety is most often addressed directly via methods that impact measurable safety metrics (e.g. safety monitors, optimization criteria, risk-aware exploration) [20]. Although the overall objective in these two fields is the same, there exist significant differences in the assumptions, problem definitions and evaluation strategies that are applied. In the following, we will briefly summarize the main lines of work and highlight existing gaps in both fields.

3.1 Safety Assurance Perspective

Traditional safety assurance approaches are based on a design time risk assessment using worst-case assumptions. For instance, ISO 26262 defines conditions for being acceptably safe with respect to functional safety. These conditions are necessary to ensure reliable and fault tolerant implementation of a system with respect to random hardware and systematic failures. However, this standard does not translate well to Deep Learning (DL) methods, since verification techniques such as white-box code coverage do not apply to DNN. As a response, several works have studied the extension of ISO 26262 to ML applications. For instance [33] analyzes the impacts that the use of ML has on the ISO 26262 safety lifecycle and gives recommendations on how to adapt the standard accordingly. The authors propose to expand the definition of hazards or the use of fault detection tools that explicitly address the ML lifecycle.

In [12], the authors argue that the main challenge for using ML in highly complex domains is arguing for an adequately low level of residual risk associated with functional insufficiencies (not functional errors) and exemplify the construction of an assurance argument for DNN classification & localization.

ISO 21448 (SOTIF) is an extension to ISO 26262 that also incorporates the concept of acceptable residual risk with respect to

[1]Throughout this work we will use the terms *novel*, *unknown* and OOD interchangeably for situations that have not occurred during the training of the agent.

acceptance criteria allocated to each safety goal of the system. This standard also includes the concept of *unknown unknowns*, i.e., input conditions that were not considered during design time but can potentially lead to harmful behavior. The standard operates on an abstract level, however. What acceptance criteria can look like for specific ML methods and how to successfully uncover unknown unknowns is only vaguely discussed.

This is addressed in [11], where a causal model is introduced, intended to aid the effective safety assurance for ML-based applications. Here, the SOTIF condition for an ML system is expressed via specific acceptance criteria and predefined assumptions on the inputs to the ML model.

The impact of uncertainties in the assurance process is further explored in [13]. Causes of uncertainty are identified in in the safety assurance of ML models with a systematic analysis of the types of asserted context, asserted evidence and asserted inference. The authors also support the position, that design time measures alone are not sufficient and need to be supplemented by convincing operation time measures. This fact is also an integral part of SOTIF, which emphasizes the need for continuous assurance.

The goal of continuous assurance is to continuously reduce uncertainties in the assurance argument to an acceptable level. This is motivated by the fact that assurance at design time can only be partial and runtime information is required to update the safety argumentation in an ongoing manner. [18] introduces through-life safety assurance as continuous process. [14] builds on this idea and proposes a method to dynamically create assurance case patterns. [4] applies the idea of continuous assurance to the dynamic safety assurance for autonomous driving applications.

Finally, there is also work that focuses on the connection between ML and safety assurance. [35] gives an overview of the current SOTA in safety assurance for ML with a focus on DNNs. This overview is structured along the phases recommended by ISO 26262 (1- req. engineering, 2- development, 3- verification, 4- validation). [27] provides an overview of practical methods for AI Safety. The authors identify several categories of insufficiencies and outline current research aiming at their detection, quantification, or mitigation. In [8] an architecture is introduced, in which a functional monitor is connected upstream of the actual ML component. This monitor decides whether the ML model takes control or whether a safe fallback policy is used. Using this architecture, the authors claim, ML components do not need to be certified, only the fault recovery system has to be. The design of the functional monitor itself is however left for future work. Importantly, none of these works address RL settings.

Notably, ISO PAS 8800 is currently under development, which aims to apply SOTIF principles to ML-based systems more generally.

3.2 Reinforcement Learning Perspective

In the field of RL, safety is mostly approached not via verifiable architectures and processes, but rather via methods that directly impact some empirically measurable safety metric, e.g. safety constraints. [10] categorizes safety claims into 3 layers. Let $c_n(s_k, a_k) \in \mathbb{R}^{n_c}$ be a set of safety constraints representing the system's safety requirements, the levels are defined as:

- **Level I**: there is no guarantee that the controller is able to adhere to the constraints but safety is encouraged via an additional term and constraint threshold d_j:
$$J_c = \mathbb{E}\left[\sum_{k=0}^{N-1} c_k(s_k, a_k)\right] \leq d_j, \tag{2}$$
- **Safety Level II** defines controllers that are able to achieve constraint satisfaction with probability $p \in (0, 1)$:
$$\Pr\left(c_k(s_k, a_k) \leq d_j\right) \geq p, \tag{3}$$
where $\Pr(\cdot)$ is a probability function.
- **Safety Level III** states that the controller must guarantee constraint satisfaction:
$$c_k(s_k, a_k) \leq d_j \tag{4}$$
for all time steps $k \in 0, ..., N$.

Methods from control theory can handle levels II and III relatively well by relying on existing models for the system dynamics. For instance, [7] integrated a Lyapunov stability-analysis criterion into the learning algorithm, to safely expand the Region of Attraction and provide strong safety guarantees. However, these approaches only work in relatively low-dimensional problems.

Most RL methods are therefore focused on level I safety, since obtaining strong guarantees is difficult in complex domains that require data-driven approaches. RL with level I safety can for instance be achieved with reward shaping [21], constrained policy optimization [1] or safety critics [37]. Another line of work aims to compensate for unsafe control actions via shielding or control-barrier functions [2, 16]. However, all these works follow a relatively narrow closed-world assumption, i.e. the resulting algorithm and its safety-relevant metrics are evaluated in the training environment. This does not resonate with most real world use cases, where environment variables can change after training. While task performance can often be compromised, it is imperative to ensure the system's safety, even under changed environmental conditions. For instance, an autonomous vehicle is allowed to slow down when facing an unknown situation but it should never crash.

Some works have discussed the use of OOD detectors as an additional additional reliability measure when deploying RL in the real world [22, 23, 30, 36]. However, the focus on safety in these works is very limited and to which extent OOD detection aids overall safety assurance arguments is not discussed. The same applies to OOD-Detection in other domains [15, 26, 29, 38].

In summary, research from the safety assurance community traditionally focused on formal guarantees and functional safety, for instance via introspective verification techniques. Most of these techniques however do not translate to DNN based systems. More recently, a lot of effort has been put into system architectures, processes and design principles that can be used to construct convincing safety arguments for ML systems. In the field of RL, most work has gone into level I safety, i.e. high autonomy but low safety. Stronger safety guarantees are currently only achievable in low dimensional environments. Furthermore, evaluation takes place almost exclusively in the training environment, impeding the possibilities of transfer to real world settings. OOD detection is a promising approach that in theory can allow for higher levels of autonomy while respecting safety more rigorously by taking over the safety responsibility from the RL agent. However, which implications OOD-detectors have towards the safety of the resulting system and how they support safety argumentation, is not yet explored. In this work we attempt to fill this gap by analyzing how concepts from the field of safety can be translated to the the field of RL and OOD-detection.

4 SAFETY CONTEXT

The motivation behind using OOD detection for RL Agents is to ensure safety in open-context settings, where the occurrence of unknown events cannot be eliminated. In the previous section we showed that open-context settings require a holistic safety framework like SOTIF, which goes beyond Design-Time-Measures. SOTIF not only introduces the concept of *unknown unknowns* as scenarios that have not occurred during training, but also emphasises the necessity of dealing with such inputs, via Operation-Time-Measure (OTM). OOD-Detection is an instantiation of an OTM, that aims to intercept unknown inputs that can potentially cause the RL agent to produce harmful outputs. Hence, OOD detection ensures that the RL agent acts in scenarios comparable to training. Consequently, we can reason about safety over all possible scenarios within a given operating domain, including unknown harmful scenarios. Overall, this defines the context and scope for OOD detection in RL applications. See Figure 1 left.

However, how to develop effective OOD measures and how to structure a convincing safety argument for these measures is not yet answered. We propose the following process, when deploying OOD detection for RL in open context scenarios:

Operational design domain (ODD) Definition Initially, the ODD needs to be specified, which is fundamental for understanding a system and its limitations. The ODD is necessary not only for OOD-Detection but also other safety arguments and informs any downstream assertions.

Understanding of potential Unknowns Next, an understanding of potentially unknown events needs to be established. This does not mean to define specific events, that can potentially occur (which is by definition not possible). Instead, the goal is to explore how unknown events can manifest in the given task and environment. This is a necessary step because unlike classification tasks, where OOD can simply be defined via classes that are absent during training, there is no clear definition of OOD for RL settings. Since there is no existing work describing how this can be accomplished, we provide several approaches for this in Section 5.

Specification of OOD Detector Given the specification of unknown events, it is possible to design and implement a specific OOD-Detector, targeted at the respective types of unknown events.

Threshold selection Depending on the selected detector, a threshold value can be set, discriminating ID and OOD samples. Depending on the application and safety requirements, this value can be set either permissive or restrictive.

Fallback Policy Finally, it is required to find a suitable fallback policy that takes over control once the OOD-Detector is triggered. In some applications, this can be a simple no-op (no-operation), in others more sophisticated policies might be necessary.

These steps are justified and evaluated as part of the *safety assurance argument*, which provides the rationale and evidence used to substantiate the direct assertion of system safety. See Figure 1-middle for an overview. However, as established by [25], the safety argument on its own may not be sufficiently convincing, given the complexity of the task, system and/or environment and needs to be complemented by a confidence argument, which in turn justifies the sufficiency of confidence in the safety argument itself. This confidence argument can address the following perspectives:

(1) **Asserted context:** confidence in the validity of context information.
(2) **Asserted solution:** confidence in the validity and integrity of evidence.
(3) **Asserted inference:** confidence in the appropriateness of the deductive logic of the argument.

When translated to the purpose of OOD detection in Reinforcement Learning, these parts attach to the assurance argument at three distinct places (see Figure 1-right):

(1) Do we understand the context of unknown situations that can occur? This requires a definition of the system boundaries (ODD) and an understanding of potential unknowns. Together these two steps inform the confidence in the asserted context.
(2) Do we have convincing (in terms of validity and integrity) evidence that our solution is capable of uncovering OOD inputs? The OOD Detector is derived from the understanding of unknown and consequently, the detector can uncover only certain types of unknowns. The confidence in the asserted evidence is therefore deduced by these two factors.
(3) Can we infer the fulfillment of higher-level goals? I.e. can the OOD detector in combination with a suitable fallback policy reduce the overall failure rate? The interplay between selected threshold (i.e. when the OOD detector gets triggered) and fallback-policy lead to the desired higher-level goal of mitigating safety violations. The confidence in inferring this consequence is grounded in these two components.

All these points need to be addressed carefully to establish a sufficient level of confidence in the safety argument itself.

5 DEFINITION OF UNKNOWN IN SEQUENTIAL DECISION MAKING PROBLEMS

As outlined in the previous section, a fundamental but missing component in safety assurance for RL is a common understanding of what is considered as *unknown* in sequential decision-making problems. In contrast to classification problems, there is no set of known classes that can be tested against. Instead, the goal is to find an optimal policy, by interacting with a training environment. Whether during testing/deployment the learned policy is presented with inputs it is not capable of dealing with, is a subtle question. Even the evaluation is intricate since there is usually no *ground truth* of the expected output for a given input (such as class labels in the classification settings). This makes defining what a trained system should or should not know much more difficult. In addition, not only one but also a sequence of inputs can be relevant to the current state of the system.

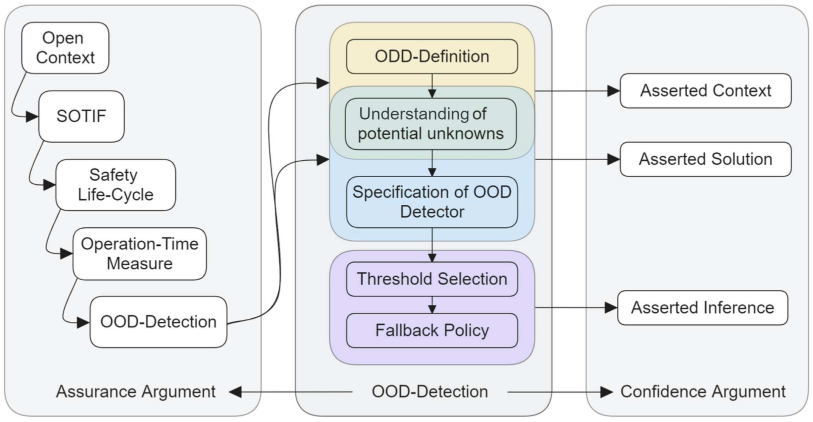

Figure 1: Safety-Context: OOD Detection is an OTM within the Safety-Life-cycle of continuous assurance frameworks (left). The process of designing OOD-Detectors constitutes several steps (middle). Each of these steps can additionaly inform a confidence argument (right)

This chapter attempts to specify the *unknown* in sequential decision making-problems. Before we present different perspectives on the problem we first discuss potential causes of the unknown.

5.1 Causes of the unknown

We discern three fundamentally different causes of the unknown.

1) Changes in the environment. Unknown situations can be a result of changes in the environment that occur after training. For instance, this can be because some aspects of the problem were not included during training or because there are external factors influencing the environment.

2) Lack of exploration. Another cause can be a lack of exploration during training. That means the agent only encounters a fraction, i.e. local region, of the state-space during training and does not sufficiently explore other parts of the environment. Therefore, from the agents perspective, inputs can be unknown, even if the environment has not changed at all. This includes events that have a very low frequency of occurrence.

3) Insufficient Learning. Also, from the agents/models perspective, events can be unknown as a result of insufficient learning during training, a lack of model capacity or catastrophic forgetting.

Causes **2)** & **3)** can be considered as *known unknowns* at the time of training since they could potentially be unveiled in the training environment. **1)** on the other hand is an *unknown unknown*. In this work we will focus on 1).

5.2 A conservative definition of unknown (train=test)

The most conservative view is to consider all conditions as unknown, that were not directly encountered during training. This can either be in terms of individual states, transitions ($\{s_t, a_t, s_{t+1}\}$-tuples) or even entire episodes. This is a very limiting view on the problem of course, as it does not allow any generalization beyond the training data. From a safety perspective, however, this is perhaps the most appealing interpretation. It allows to make strict guarantees on the behavior of the agent since it is already known from training.

5.3 Similarity based definitions of unknown

To enable generalization beyond training, it is necessary to stretch the interpretation of known scenarios to situations that are *similar* to those encountered during training. This shifts the question to what is considered an appropriate measure of similarity between situations. Let a situation be defined as an indefinite sequence of states and actions. The simplest way to quantify the similarity between situations is via measures over single states.

S1: Single states Let s and s' be two single state vectors. The most basic measure of similarity between states is via a distance metric $d : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ in the state-vector space, such as the p-norm:

$$sim(s, s') := -d(s, s') := - \mid\mid s - s' \mid\mid_p, \tag{5}$$

where $\mid\mid s-s' \mid\mid_p = \left(\sum_{i=1}^{M} \mid s_i - s_i' \mid^p\right)^{\frac{1}{p}}$ and M is the dimensionality of the state-space. Other measures such as the Cosine- or Jaccard-similarity apply equally. See Figure 2-left for a visualization.

S2: Trajectories S1 extends to trajectories (or episodes) as

$$sim(\tau, \tau') := \sum_{t}^{T} -d(\tau_t, \tau_t'). \tag{6}$$

with two trajectories of the form $\tau = ((s, a)_0, (s, a)_1, \ldots (s, a)_n,)$. While this definition already allows for some generalization, it does not capture any semantic information. It also doesn't scale well with the dimensionality of the problem space, especially if time is considered. For a visualization, see figure 2. The right-hand side pink trajectory lies further away from the training data, but has the same shape. The other pink trajectory is closer to the training data and would thus be considered more similar, although it has an entirely different shape. Evidently, this notion of similarity quickly reaches its boundaries and does not translate to high-dimensional problems.

S3. Process/distribution To tackle the curse of dimensionality, the training distribution can be modeled explicitly, i.e., the MDP $\mathcal{M}$

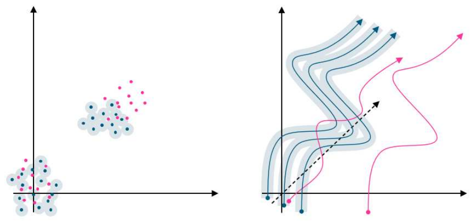
Figure 2: Visualization of states visited during training (blue) and deployment (pink). On the left, time is not considered and the states are spread out along their two dimensions. On the right, time is considered and the states form a sequence/trajectory. The light blue shading indicates a small similarity margin in the vector space (e.g. euclidean norm).

that creates the training distribution. If the MDP can be described by a simple function, for instance, a Gaussian distribution, the similarity of the currently observed MDP to the training MDP can be measured via the density of the observed samples under the training distribution:

$$sim(\mathcal{M}, s') := p\left(s' \mid \mu_{\mathcal{M}}, \Sigma_{\mathcal{M}}\right) \tag{7}$$

or via the KL divergence of training and deployment distribution

$$sim(\mathcal{M}, \mathcal{M}') := D_{KL}\left(\mathcal{M} \mid \mathcal{M}'\right). \tag{8}$$

However, MDPs are usually more complex than a simple function. [23] suggest that all components of the MDP can be subject to some change, $(\mathcal{S}, \mathcal{A}, r, f, \mu_0)$, and can thus also be measured. Though, approximating all these aspects solely via interaction is typically intractable. A surrogate for this is modeling the training MDP with a universal function approximator f_θ, such as a neural net as in [22]. We can then compare predictions to observed states:

$$sim(\mathcal{M}, \mathcal{M}') := -d\left(f_\theta(s'_t, a'_t), s'_{t+1}\right). \tag{9}$$

S4: Context If some parameters of an environment change (e.g. friction parameters of a robot), technically, the MDP has changed ($\mathcal{M}_{train} \neq \mathcal{M}_{deploy}$). Although, the fundamental process is still the same. Such changes can be isolated into a context variable C, which contains certain properties of an MDP. This can be modeled via a Contextual-MDP [24] as $(C, \mathcal{S}, \mathcal{A}, \mathcal{M}(c))$, where $\mathcal{M}(c)$ is a function mapping a context $c \in C$ to an MDP $\mathcal{M}(c) = (\mathcal{S}, \mathcal{A}, r^c, f^c, \mu_0^c)$.

S5: Semantic similarity While the context can be interpreted as a means of isolating domain information, it does not specifically capture semantics. Semantic information refers to "meaning", rather than just correlation. For instance, if the inputs are images, the semantics could be object classes, rather than pure pixel values. However, this is highly task specific. To the best of our knowledge, there is no universal method to measure semantic similarity.

S6: Reward/value function Another formulation for measuring the similarity between training and deployment situations is via the reward/value functions. The assumption is that two states are similar if their (discounted) future reward is similar. However, the reward signal is usually not accessible during deployment.

S7: The task Similarity can also be focused on the task itself. For instance, if the task is for a robot arm to pick up boxes or to throw boxes, the encountered states are totally different, although the environment has not changed. The reward function in a way also captures this - it reinforces certain behavior.

All these perspectives are an attempt to improve our understanding of the problem. This is by no means an exhaustive list. Nonetheless, it can serve as a starting point when designing OOD detectors and eventually structure a safety case for RL agents. Depending on the perspective, OOD detectors have to operate on certain abstraction layers of the problem, which in turn means that they can only detect certain types of anomalies. It therefore requires careful consideration on which notion to operate on, when designing and testing OOD detectors.

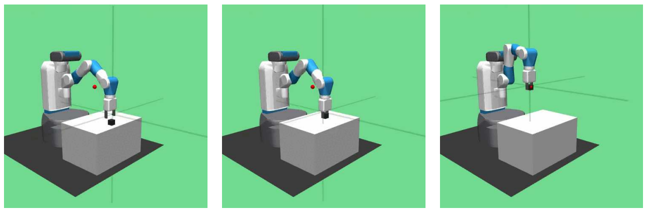
Figure 3: Exemplary use-case: *FetchPickAndPlace* - environment [17]. The goal for the robot is to pick up a square box from a table and move it to a desired goal location (red point).

6 EXPERIMENTS

In this section, we translate the above findings into a practical example. Our goal is to illustrate the relationship between the specification of unknown, the design of OOD Detectors and the choice of a fallback Policy

For this, we evaluate three concrete OOD-Detectors in an exemplary use case: the pick-and-place task from the gymnasium robotics test suite [17]. Consider a multi-joint 7-DoF robot arm, with a two-fingered parallel gripper attached to its end-effector. The task for the robot is to pick up a box from a table using its gripper and move it to a desired goal location (see Figure 3). The initial location of the box and the goal are randomized over a 2-D/3-D space respectively. The observation consists of kinematic information of the box and the gripper, as well as the goal location ($\mathcal{S} \in \mathbb{R}^{25}$). An action represents the Cartesian displacement of the robot's end effector ($\mathcal{A} \in \mathbb{R}^4$).

In a closed-world setting, a policy can be trained via a state of the art RL algorithm[2], that succeeds at this task around 99% of the time. *Closed-world* means, that the parameters during training are exactly the same as during deployment. Hence, only the goal- and box location change at the beginning of each episode, while all other parameters, such as box mass, friction of the robot joints, etc. stay exactly the same.

Training Initially, the RL policy is trained until convergence in the training environment (the original pick-and-place environment from [17]). After that, we freeze the policy parameters and generate 100 episodes from the training environment as training data for the OOD-Detectors.

Test Data For exemplary testing we construct a series of OOD scenarios. These are variations of the original training environment, with one of the following semantic perturbations:

[2]We use TQC [28] with HER [3] and train for 3M steps with sparse rewards.

	Reward	Success	Fail	Critical
ID Env	*-6.96*	*99*	*1*	*0*
Big Box	-48.52	14	86	52
Heavy Box	-19.41	82	18	30
Sphere	-34.03	56	44	36
Box Position	-20.20	89	11	12
Moving Box	-47.02	44	56	68

Table 1: High level experiment statistics for 100 episodes in each environment. *Reward* is the average cumulative reward over episodes. *Success*, *Fail* and *Critical* are absolute counts.

- **Big Box:** the size of the Box is increased (2x in each dimension).
- **Heavy Box:** the weight of the Box is increased (20x).
- **Sphere:** The box is replaced with a sphere.
- **Box Position:** The box is initialized at extreme $\{x, y\}$- positions, close to the edges of the table.
- **Moving Box**: The box moves with a constant velocity in y-direction (0.25 ms^{-1}).

We then generate 100 episodes from each disturbed environment by applying the (static) RL policy. This constitutes 5 different test sets (1 for each disturbed environment). On strict terms, these perturbations turn into known unknowns as soon as they are defined. For the purpose of this evaluation, we argue it is reasonable to consider them as exemplary unknown unknowns. All these perturbations were not considered during training and can thus be regarded as unknown events. As mentioned above, evaluation on truly unknown unknowns is otherwise not possible.

High-level results On a high level, we are interested in both performance and safety metrics. As performance relevant metric we consider the number of times the policy succeeds/fails to move the box to the goal as well as the cumulative reward (-1 for each time step the block hasn't reached its final target position, and 0 if the block is in the final target position). As safety critical metric we consider all episodes, in which the box drops to the floor or the table from more than 10cm of height. The resulting high-level statistics are summarized in Table 1. In the unmodified ID environment, the policy succeeds to move the box to the goal location in 99 of the 100 episodes. It fails once, but without a safety-critical event. This is entirely different in OOD cases, where the critical rate gets as high as 68%.

OOD Detectors We evaluate three different types of OOD detectors. These detectors directly derive from the first three definitions of *unknown* from Section 5.3:

- $\mathcal{S}$**-KNN**: measures similarity via L2-norm of single states at each time-step (builds on **S1**):

$$sim(s_t, s_t') := - \mid\mid s_t - s_t' \mid\mid_2$$

- **LSTM**: measures similarity via the sequential n-step prediction error (builds on **S2**):

$$sim\left(s_{t:t+n}, s'_{t:t+n}\right) := - \mid\mid f_\theta(s_t, h_{t-1}) - s'_{t:t+n} \mid\mid_2$$

- **PEDM[22]:** measures similarity via approximation of the MDPs at each timestep (builds on S3):

$$sim(\mathcal{M}_t, \mathcal{M}_t') := - \mid\mid f_\theta(s_t, a_t) - s'_{t+1} \mid\mid_2 .$$

ID and OOD labels are assigned at each timestep as:

$$\hat{y} = \begin{cases} ID, & \text{if } sim(\cdot) < \vartheta \\ OOD, & \text{otherwise} \end{cases} \tag{10}$$

where ϑ is a threshold score. The KNN detector uses the training data as an index for comparison during inference. LSTM and PEDM are fitted on the training data until convergence.

The evaluation procedure for the detectors is not trivial. Fundamentally we take two different approaches to this: 1) Evaluation via ability of detecting disturbances and 2) Evaluation via detection of safety-critical events. Both are described in the following. [3]

6.1 Evaluation via detection of disturbance

The evaluation via the detection of disturbances is based on evaluation procedures found in the ML literature. That is, we assume access to high-level information about the presence of disturbances in a given episode and label data accordingly - all steps from the ID environment are labeled as ID and all steps from the OOD environments as OOD. This results in a balanced test set for each disturbance. This allows us to compute theoretical evaluation metrics such as the AUROC (area under the receiver operator characteristic), AP (average precision), or FPR95 (false positive rate at 95 % recall). These are the most prevalent metrics for measuring the performance of binary classifiers in the ML literature. The results are summarized in Table 2.

All detectors achieve relatively high scores in *Big Box*, *Sphere*, and *Moving Box*. For *Heavy Box* and *Box-Position*, the scores are visibly lower. For the former, PEDM is marginally better while for the latter, the KNN is superior. The reason for this lies in the principles the detectors operate on. $\mathcal{S}$-KNN only considers single states so it does not incorporate any dynamics. It can however detect more extreme single states. PEDM on the other hand only operates on the dynamics of the environment. These are still somewhat predictable, even if the initial position has a small offset. The change in the box mass on the other hand results in unpredictable dynamics, which in turn leads to better detections with PEDM. Overall, and according to these metrics, the detectors present comparable performance.

Implications on safety are, however, difficult to derive from these metrics alone. To understand this, the perspective needs to be shifted to safety-critical events.

6.2 Evaluation via safety-critical events

The basic objective of an OOD-detector is to trigger *before* a safety-critical event. Otherwise, intervention is not possible. To evaluate this property, we measure the time between detection and occurrence of safety-critical events as buffer time t_b. Let t_c be the time-step at which a safety-critical event occurs, and t_d the time-step at which a detector infers an OOD label. The buffer time is defined as:

$$t_b = t_c - t_d \tag{11}$$

To evaluate the detectors in this manner, we also need to specify a fixed threshold on the detector score which discriminates between ID and OOD samples (see eq. (10)). Since we consider OOD events as *unknown unknown*, we do not have access to any OOD data during

[3]The environment test suite as well as the code for all experiments is available at https://github.com/FraunhoferIKS/safety-ood-rl.

	AUROC			AP			FPR95		
	S-KNN	**LSTM**	**PEDM**	**S-KNN**	**LSTM**	**PEDM**	**S-KNN**	**LSTM**	**PEDM**
Big Box	0.99	0.99	0.99	0.99	0.98	0.99	0.04	0.04	0.04
Heavy Box	0.76	0.77	0.78	0.80	0.79	0.81	0.86	0.76	0.87
Sphere	0.97	0.98	0.98	0.98	0.97	0.98	0.23	0.14	0.08
Box Position	0.83	0.73	0.77	0.86	0.75	0.77	0.76	0.87	0.73
Moving Box	0.95	0.98	0.98	0.97	0.97	0.98	0.28	0.11	0.07
Avg.	0.90	0.89	0.90	0.92	0.89	0.91	0.43	0.39	0.36

Table 2: *Evaluation metrics via detection of disturbance.* AUROC, AP and FPR95 are calculated based on step-wise labels from 100 episodes for each environment. All steps from the ID environment are labeled ID, all steps from the perturbed environments as OOD, disregarding safety-critical events or performance loss.

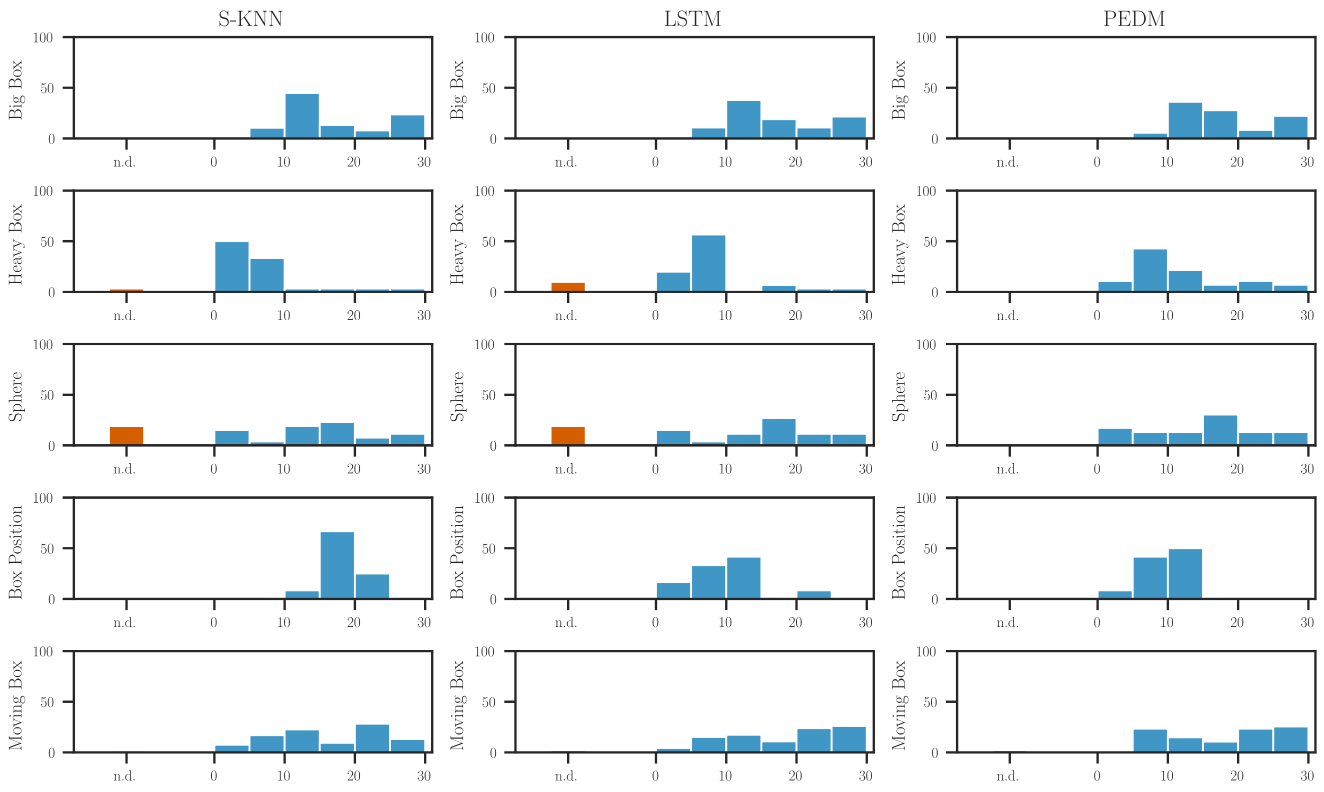

Figure 4: *Evaluation via safety-critical events.* *x-axis:* Safety buffer (time between OOD detection and safety-critical event). *y-axis:* percentage of timesteps where the detector raised an alert x timesteps before the critical event (blue bars). If the alert is raised after the critical event, it counts as not detected (n.d., orange bar). The more this distribution concentrates on the right-hand side, the better.

training. This means we cannot use OOD data for thresholding the detectors. Therefore, we also generate a validation set from the ID environment and threshold the detectors with the TPR-β method, such that the 99th percentile of the validation data is classified as ID - a common choice in the ML literature [9].

The results of this procedure are visualized in Figure 4. An analysis of the results leads to slightly different conclusions, compared to the previous section. For instance, KNN and LSTM are comparable with PEDM in *Sphere*, considering AUROC scores from above. Analyzing detection times, however, LSTM and KNN infer an OOD label often only after the safety-critical event (orange bar), essentially defeating the purpose of an online safety monitor. In *Box Position*, on the other hand, KNN is most effective, inferring OOD labels with significantly larger t_b.

6.3 Evaluation via time-critical labels

The time between OOD detections and actual safety-critical events is an important measure. But, how much buffer time t_b is required, s.t. safety-critical events can be prevented via intervention?

	critical event	**prevention possible**	**avg. req. buffer**
Big Box	52	51	12.64
Heavy Box	30	30	18.86
Sphere	36	32	16.34
Box Position	12	11	9.90
Moving Box	68	27	20.22

Table 3: *Theoretically required time before failure.* Number of episodes with a critical event and amount of critical events that could have been prevented if a fallback policy is activated early enough. The avg. required buffer time is the average time before a safety-critical event after which a violation becomes inevitable, even when following a *safe* fallback policy.

	TPR			FPR		
	S-KNN	LSTM	PEDM	S-KNN	LSTM	PEDM
Big Box	0.83	0.94	0.98	1.00	0.98	0.98
Heavy Box	0.10	0.13	0.43	0.40	0.54	0.56
Sphere	0.58	0.69	0.81	0.97	0.97	0.98
Box Position	0.92	0.67	0.75	0.86	0.70	0.52
Moving Box	0.18	0.32	0.29	1.00	1.00	1.00
Avg.	0.52	0.55	0.65	0.85	0.84	0.81

Table 4: Evaluation using theoretically required time buffer before safety-critical event as OOD label ground-truth.

To answer this question we repeat the experiments from above, reconstructing the initial conditions of each episode exactly. For each episode with a safety-critical event, we start at t_c and work our way backwards, intervening the *RL* policy at timestep $t_I \in \{t_{c-1}, t_{c-2}, ..., t_0\}$ with an intervention policy π_I. This results in trajectories $\tau := (\tau_{0:I} \sim \pi_{RL}, \tau_{I:T} \sim \pi_I)$. As a fallback policy, we simply apply no-op (not moving any of the robot actuators). We do this until we find a timestep t_I from which onward applying the fallback policy does not lead to a safety-critical event. The results are summarized in Table 3. Notably, the buffer time required before critical events is heavily dependent on the type of perturbation.

For each episode, we can now reevaluate the detectors with this information, according to the following scheme. If the detector produces an OOD label before the last possible timestep to prevent a given safety-critical event, this counts as a true positive. If the label occurs too late or not at all, this counts as a false negative. True negatives and false positives are calculated in the opposite way. The resulting true-positive and false-positive rates are summarized in Table 4. Comparing these numbers to the results from Section 6.1, it is apparent that considering both realistic thresholding and the required safety buffer in the evaluation results in a much more difficult task. Generally, the TPRs are relatively low but they also vary drastically, depending on the type of disturbance. Some disturbances are more difficult to detect in time than others. Given that the observations the agent/detector receives only contains kinematic information about the robot and the box, some disturbances only become apparent after some interaction of the robot with the box, only when considering several consecutive timesteps or when modeling system dynamics. This is prominent for *Sphere* and *Heavy Box*, where PEDM has significantly higher TPR scores than the other detectors since it is the only detector to consider system dynamics. Single extreme states on the other hand such as the *Box Position* are better detected with a simple KNN.

6.4 Confidence in the assurance argument

Coming back to the safety argumentation, there are a few observations we can make from these experiments.

1. It is important to understand what types of unknowns are likely to occur in a given domain. This understanding is paramount to the design of effective OOD-Detectors, as well as establishing confidence in the Asserted Context.

2. It is important to measure the correct metrics for a given task. Confidence in achieving the safety requirements can only be established if there is a convincing argument that the collected evidence is appropriate (Asserted Solution). Here, we only provide an illustration of potential pitfalls. Future work shall explore how to gather evidence for OOD Detectors in a way that helps when constructing a convincing safety argumentation at design time and with continuous assurance.

3. It is important to consider the fallback policy. After all, OOD Detection is useless if there is no fallback mechanism that can keep the system in a safe state. Confidence in the argumentation strategy (Asserted Inference) depends on the above discussed metrics as well as this fallback policy.

7 CONCLUSION & OUTLOOK

In this paper we showed that one of the main challenges for using RL in safety-critical real-world applications is arguing for the safety of the intended functionality in light of unknown events. This challenge already starts with understanding what constitutes an *unknown event*. This paper is an attempt at finding such an understanding, that is beneficial for both the safety and the ML communities.

We showed that open-context settings such as real-world RL applications require continuous assurance, since classical verification & validation approaches do not apply. We identified OOD detection as a key component in such frameworks that require OTM for ensuring safety at runtime. When developing OOD detectors it's crucial to account for the ways novelties can emerge, impacting detector effectiveness and safety claims. In this work, we presented several different views that aid this process. Incorporating these findings into a series of experiments, we analyzed the contribution of OOD detection, leading to the following observations.

OOD detection is a small but no less important ingredient in the safety assurance of RL systems. It can be used as an operation-time measure intercepting inputs that could otherwise lead to hazardous behavior of the RL agent. What OOD detection does not provide is an accurate answer to whether the detected situations pose an actual safety threat or not. In other words, it is an overly sensitive but not very specific measure, prone to false positives. Whether unknown states are potentially hazardous is highly task dependent and requires further assumptions. If and how this causal relationship can be established will be investigated in future work.

We also showed that the evaluation scheme itself plays an important role when arguing about the effectiveness of OOD detectors in the context of RL. In order to do this, we have contrasted different approaches from both the ML and safety perspectives. We have found that a pure ML-perspective serves as a theoretical measure to compare different detectors on an abstract level but provides only limited insights about safety in downstream applications. It is therefore important to also consider safety relevant metrics such as the time between the occurrence of critical events and the actual detection time. Future work can also explore further evaluation schemes, considering more complicated or more realistic scenarios.

It is also important to note that in most cases, even a very good OOD detector will not have 100% accuracy. In complex domains, there will always be unknown inputs, the system cannot detect as such. It is therefore necessary to embed the RL agent together with the OOD detector into an adaptive architecture. This encompasses continuous learning from both the training and the operation domain and updating the components continuously.

ACKNOWLEDGMENTS

This work was funded by the Bavarian Ministry for Economic Affairs, Regional Development and Energy as part of a project to support the thematic development of the Institute for Cognitive Systems.

REFERENCES

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. In *International conference on machine learning*. PMLR, 22–31.

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. *Advances in neural information processing systems* 30 (2017).

[4] Erfan Asaadi, Ewen Denney, Jonathan Menzies, Ganesh J Pai, and Dimo Petroff. 2020. Dynamic assurance cases: a pathway to trusted autonomy. *Computer* 53, 12 (2020), 35–46.

[5] Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. 2020. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature* 588, 7836 (2020), 77–82.

[6] Richard Bellman. 1957. A Markovian decision process. *Journal of mathematics and mechanics* (1957), 679–684.

[7] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. 2017. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems* 30 (2017).

[8] Alessandro Biondi, Federico Nesti, Giorgiomaria Cicero, Daniel Casini, and Giorgio Buttazzo. 2019. A safe, secure, and predictable software architecture for deep learning in safety-critical systems. *IEEE Embedded Systems Letters* 12, 3 (2019), 78–82.

[9] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[10] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), 411–444.

[11] Simon Burton. 2022. A causal model of safety assurance for machine learning. *arXiv preprint arXiv:2201.05451* (2022).

[12] Simon Burton, Lydia Gauerhof, and Christian Heinzemann. 2017. Making the case for safety of machine learning in highly automated driving. In *Computer Safety, Reliability, and Security: SAFECOMP 2017 Workshops, ASSURE, DECSoS, SASSUR, TELERISE, and TIPS, Trento, Italy, September 12, 2017, Proceedings 36*. Springer, 5–16.

[13] Simon Burton and Benjamin Herd. 2023. Addressing uncertainty in the safety assurance of machine-learning. *Frontiers in Computer Science* 5 (2023), 1132580.

[14] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2017. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1039–1069.

[15] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[16] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. 2019. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 3387–3395.

[17] Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. 2023. *Gymnasium Robotics*. http://github.com/Farama-Foundation/Gymnasium-Robotics

[18] Ewen Denney, Ganesh Pai, and Ibrahim Habli. 2015. Dynamic safety cases for through-life safety assurance. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 587–590.

[19] Ian Fox, Joyce Lee, Rodica Pop-Busui, and Jenna Wiens. 2020. Deep Reinforcement Learning for Closed-Loop Blood Glucose Control. In *Proceedings of the 5th Machine Learning for Healthcare Conference*. PMLR, 508–536.

[20] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

[21] Marek Grzes. 2017. Reward shaping in episodic reinforcement learning. (2017).

[22] Tom Haider, Karsten Roscher, Felippe Schmoeller Roza, and Stephan Günnemann. 2023. Out-of-Distribution Detection for Reinforcement Learning Agents with Probabilistic Dynamics Models. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems*.

[23] Tom Haider, Felippe Schmoeller Roza, Dirk Eilers, Karsten Roscher, and Stephan Günnemann. 2021. Domain Shifts in Reinforcement Learning: Identifying Disturbances in Environments.. In *AISafety@ IJCAI*.

[24] Assaf Hallak, Dotan Di Castro, and Shie Mannor. 2015. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259* (2015).

[25] Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. 2011. A new approach to creating clear safety arguments. In *Advances in Systems Safety: Proceedings of the Nineteenth Safety-Critical Systems Symposium, Southampton, UK, 8-10th February 2011*. Springer, 3–23.

[26] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).

[27] Sebastian Houben, Stephanie Abrecht, Maram Akila, Andreas Bär, Felix Brockherde, Patrick Feifel, Tim Fingscheidt, Sujan Sai Gannamaneni, Seyed Eghbal Ghobadi, Ahmed Hammam, et al. 2022. Inspect, understand, overcome: A survey of practical methods for ai safety. In *Deep Neural Networks and Data for Automated Driving: Robustness, Uncertainty Quantification, and Insights Towards Safety*. Springer International Publishing Cham, 3–78.

[28] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. 2020. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*. PMLR, 5556–5566.

[29] Kwei-Herng Lai, Daochen Zha, Junjie Xu, Yue Zhao, Guanchu Wang, and Xia Hu. 2021. Revisiting time series outlier detection: Definitions and benchmarks. In *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 1)*.

[30] Robert Müller, Steffen Illium, Thomy Phan, Tom Haider, and Claudia Linnhoff-Popien. 2022. Towards Anomaly Detection in Reinforcement Learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 1799–1803.

[31] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. 2019. Deep Dynamics Models for Learning Dexterous Manipulation. *arXiv:1909.11652 [cs]* (Sept. 2019).

[32] Matt Osborne, Richard Hawkins, and John McDermid. 2022. Analysing the Safety of Decision-Making in Autonomous Systems. In *Computer Safety, Reliability, and Security: 41st International Conference, SAFECOMP*. Springer, 3–16.

[33] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki. 2017. An analysis of ISO 26262: Using machine learning safely in automotive software. *arXiv preprint arXiv:1709.02435* (2017).

[34] Walter J Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E Boult. 2012. Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence* 35, 7 (2012), 1757–1772.

[35] Gesina Schwalbe and Martin Schels. 2020. A survey on methods for the safety assurance of machine learning based systems. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*.

[36] Andreas Sedlmeier, Thomas Gabor, Thomy Phan, Lenz Belzner, and Claudia Linnhoff-Popien. 2019. Uncertainty-based out-of-distribution detection in deep reinforcement learning. *arXiv preprint arXiv:1901.02219* (2019).

[37] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. 2020. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603* (2020).

[38] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. 2021. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334* (2021).