IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 20, 2020, accepted August 31, 2020, date of publication September 11, 2020, date of current version September 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3023394

Reinforcement Learning Interpretation Methods: A Survey

ALNOUR ALHARIN, (Member, IEEE), THANH-NAM DOAN, AND MINA SARTIPI, (Senior Member, IEEE)
Department of Computer Science and Engineering, The University of Tennessee at Chattanooga, Chattanooga, TN 37403, USA
Corresponding author: Alnour Alharin (dyc881@mocs.utc.edu)

This work was supported in part by NSF-US Ignite under Grant 1647161.

ABSTRACT Reinforcement Learning (RL) systems achieved outstanding performance in different domains such as Atari games, finance, healthcare, and self-driving cars. However, their black-box nature complicates their use, especially in critical applications such as healthcare. To solve this problem, researchers have proposed different approaches to interpret RL models. Some of these methods were adopted from machine learning, while others were designed specifically for RL. The main objective of this paper is to show and explain RL interpretation methods, the metrics used to classify them, and how these metrics were applied to understand the internal details of RL models. We reviewed papers that propose new RL interpretation methods, improve the old ones, or discuss the pros and cons of the existing methods.

INDEX TERMS Reinforcement learning, machine learning, interpretability, interpretation, survey.

I. INTRODUCTION

Reinforcement learning algorithms achieved a good performance on multiple domains. However, our inability to explain and justify their decisions, makes it harder to deploy RL systems in some critical fields such as healthcare, where interpretability is necessary [1].

Researchers proposed various RL interpretation methods and applied them to multiple applications. While the broad objective of RL interpretation is to make RL policies more understandable, each method has its own special purposes, sets of applicable problems, limitations, and challenges. Nonetheless, RL interpretability lacks the same variety and depth of survey papers, when compared to ML interpretability, despite the fact that some researchers made a good effort in reviewing and categorizing the work related to their proposed approaches such as the work done by Nikulin *et al.* [2] in classification of saliency maps.

The goal of this paper is to list, classify and compare different methods used to interpret RL. We consider the interpretation methods of both classical and deep RL. There are several papers on RL interpretation, and this paper will help with organizing, summarizing, and adding to those existing survey works. This paper will also provide engineers and researchers using RL means to compare different interpretability approaches to help them choosing the most appropriate method for their RL systems. Finally, it will help us to identify the gaps, open challenges, and future directions so that new researchers find it easier to contribute to this field.

The rest of this paper is organized as follows: Section II summarizes the related work. Section III illustrates our relevancy criteria in selecting papers. Section IV gives a brief explanation to the concepts used in this field. Section V gives a high level overview of the RL interpretation methods. Section VI lists and explains different categorization metrics used to classify RL interpretation methods, and how these metrics can be applied. Section VII explains RL interpretation methods in more details. Finally, Section VIII gives the conclusion of the paper.

II. RELATED WORK

Interpretability models are challenging to evaluate. One of the main reasons is the ambiguity of the used terms. Some authors focused on defining these terms and illustrating the differences between the similar ones. For example, Weller [3] discussed *transparency*, its different meanings, motivations and related challenges. They also showed that transparency is not always useful, as sometimes it might make the system vulnerable to abuse and manipulation by outside intruders.

The associate editor coordinating the review of this manuscript and approving it for publication was Ioannis Schizas.

Similarly, Gilpin *et al.* [4], [5] tried to distinguish between interpretability and explainability, and proposed some principles to evaluate ML interpretation methods. They used the *explanation* with a meaning of *an answer to a why-question* and interpretability of the system to be its *ability to provide understandable explanations to the users*. They also defined the *completeness* of the generated explanations to be their ability to describe the system as accurate as possible. In addition, Mohseni *et al.* [6] showed that differing goals is another source of confusion and ambiguity. According to the authors [6], having different objectives can lead to building different evaluation criteria. To support their claim, the authors mapped the goals of people from different domains (e.g., ML, psychology, etc.) to the corresponding evaluation methods and proposed a framework to link between the interpretability evaluation method and the goals of the corresponding group. On the same track, Guidotti *et al.* provided a theoretical framework that divides the interpretability problem into three parts: model explanation, outcome explanation, and model inspection [7]. Then, they provided formal definitions for each part, and applied it on the interpretation methods they reviewed.

Several researchers summarized, classified, and evaluated the interpretability of machine learning as general. For example, Molnar [8] explained the importance of machine learning interpretability, classified different ML interpretation methods by utilizing different metrics, and provided examples of their application on three selected datasets.

We also know of some efforts of building theoretical frameworks for interpretability evaluation. For instance, Bibal and Frénay [9] proposed a unified and structured framework to answer the two questions of ''what is interpretability?'' and ''how to measure it''. Similarly, James Murdoch *et al.* [10] proposed a predictive, descriptive, relevant (PDR) framework for ML accuracy evaluation that takes the interpretability part into account. Their method evaluates ML systems based on their predictive accuracy, descriptive accuracy (fidelity of the interpretation method as explained in Section VI-F), and relevancy (how much domain-relevant insights are provided by the model), instead of only using predictive accuracy in the typical ML evaluation pipeline. We should note that relevance in this context is measured by humans through surveys.

In addition to working on the theoretical principles of interpretability and their evaluation metrics, some researchers classified and evaluated the existent interpretability techniques. For example, Biran and Cotton [11] classified ML interpretation methods into *prediction interpretation* and *interpretable models* and made a survey based on this. Adadi and Berrada [12] provided an overview for the different approaches in ML interpretability by analyzing more than 280 papers in the field.

Besides the papers that classify the interpretation methods regardless of the application domain, there are also specialized work such as [13] which focuses on healthcare applications. The authors [13] discussed the challenges facing ML interpretability in healthcare and the different parameters that should be considered in their application. Similarly, Zhang and Zhu [14], [15] focused on the interpretation of CNNs. They studied different methods for CNN visualizations, and the methods used to understand the CNN layers representations and their internal filters (namely filter interpretability [16] and location instability [15]).

III. RELEVANCY CRITERIA

We consider a paper relevant for our survey if it proposes a new method of RL interpretation or improves an old one. We also consider papers that discuss a known RL interpretation method (e.g., states its advantages and drawbacks), or study the conditions or metrics that RL interpretation methods should satisfy. We collected the papers mainly from Google Scholar search engine, and we also followed the citations and references of each paper looking for more relevant papers.

IV. BACKGROUND

Machine Learning (ML) systems learn patterns automatically from data without writing an explicit algorithm [17]. Instead, they rely on picking a model that best fits the data from a hypothesis set. A more formal way to define ML systems is the one proposed by Mitchel [17] *''A machine is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E''*. Reinforcement Learning (RL) is one of the emerging sub-fields of ML that achieved outstanding performance in different domains such as robotics [18], self driving cars [19], health informatics [20], beating humans in complex games such as Go [21], and playing Atari games [22]. It studies the systems that can be represented with an agent which interacts with a dynamic environment through taking different actions in order to maximize its cumulative reward [23]. The agent has a specific state that belongs to a set of limited legal possible states known as the state space. After choosing an action from the set of legal actions (actions set), the agent receives a reward (a numeric value) that depends on the chosen action and its current state. State space and actions set are determined by the environment. RL systems might have more than one agent, and in this case we call the system a multi-agent RL (MARL). However, we will focus mainly on the interpretability of single-agent RL systems. Generally, RL systems can be formulated by using the Markov Decision Process (MDP), and the more generic form of it: Partially Observable Markov Decision Process (POMDP) which will be explained in sections IV-A and IV-B respectively. Then after that, we will speak about some important concepts in RL such as policy in Section IV-C, types of RL systems in Section IV-D, and different RL learning algorithms in Section IV-E.

A. MARKOV DECISION PROCESS (MDP)

MDP is a mathematical framework used to study decision problems, where the resultant state of any decision does not depend on the history of the previous states. Instead, it depends only on the current state and the action of the agent. In addition, there is also an uncertainty in determining the final decision. i.e., agent decisions are partially random. This process can be described mathematically as follows:

The agent is at state $s \in S$. It can choose any allowed action $a \in A$. The legal actions are a subset of A, since not all actions are allowed from a given state. The set of these legal actions can be described as $A(s) \subset A$. Taking an action a changes the agent state from a current state s to a new state s' through the transition function T, and subsequently it receives a reward $R_a(s, s') \in R$ that depends on the previous state s, the new state s' and the action a. Mathematically, MDP is a tuple of (S, A, T, R) where:

1) S: set of all possible states (state space).
2) A: set of all possible actions.
3) $T_a(s, s') : A \times S \times S \rightarrow [0, 1]$: a transition probability function that returns the probability of moving from state s to s' when performing action a.
4) $R_a(s, s')$: the reward that the agent receives when action a leads from s to s'.

B. PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

In Partially Observable Markov Decision Processes (POMDP), the state is not fully observable by the agent. Mathematically we can define POMDP as a tuple of $(S, A, T, R, \Omega, O, \gamma)$ where:

1) S, A, T, R: the same meanings of MDP in Section IV-A.
2) Ω: set of all observations.
3) O: set of observation probabilities.
4) $\gamma \in [0, 1]$: the discount factor that is used to penalize the future rewards compared with the immediate reward.

C. POLICY AND VALUE FUNCTION

As mentioned above, the agent needs to choose an action from the actions set. This action is chosen is based on the agent's policy, which can be defined as the strategy that the agent follows to choose the actions. It is a mapping between the agent state and the chosen action at that state. A policy can be either deterministic or non-deterministic. In a deterministic policy, the agent takes the same action at the same state, and its policy π can be described by the equation: $\pi(s) = a$, where s is the agent state and a is the action. For non-deterministic policies, the mapping is formulated in a probabilistic format, where we have a probability for each action, given the agent state as follows: $\pi(a|s) = Pr(a_t = a|s_t = s)$ where $\pi(a|s) \in [0, 1]$.

In addition to the policy, one of the most important concepts in RL is value function. It is a way to measure how good is to be in a specific state under a given policy (π). Different policies assign different values for the same state. The value function for a state s, is the sum of the expected rewards of the agent starting from s. This can be written as follows:

$$V^{\pi}(s) = \mathbb{E}(\sum_{i=1}^{T} \gamma^{i-1} r_i), \tag{1}$$

where $\gamma \in [0, 1]$ is the discounting factor, $r_i \in R$ is the reward at time i, T is the number of time steps, π is the policy, and s is the agent state.

D. TYPES OF RL SYSTEMS

Generally, RL models can be either value based, policy based, or model based. In the value based RL, we first calculate the value function for each state as stated in Equation 1, and then we use these values to evaluate the policy. In contrast, we evaluate and improve the agent's policy iteratively in policy based RL. Note that both these methods do not include assumptions about the transition function (the model of the environment), and hence they are called *model-free RL*. Moreover, we can also combine these approaches in a new type called *actor-critic* RL, where the *critic* calculates the value function while the *actor* updates the policy using the values calculated by the critic.

On the other side, model based-RL includes a model about the environment (a transition function). This function gives the probability of moving from a state to another given the agent action. We should note that the agent state might either be fully or partially observable. Fully observable environments are modeled by MDP while partially observable environments are modeled by POMDP.

E. LEARNING ALGORITHMS

The optimal policy is the one that maximizes the long term expected rewards as described by Equation 1. This maximization can be formulated by the Bellman equation which gives the optimal policy for a state s as follows [24]:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s' \in S} T(s, a, s') V^{\pi^*}(s'), \tag{2}$$

where V^{π} is the value function under the policy π. Different methods are used to optimize this problem, such as value iteration, policy iteration, temporal difference learning, Monte Carlo methods, and $Q-$learning [24]. In the next sections, we will give brief overviews of these methods.

1) VALUE ITERATION

In this approach, we first initialize the value functions with random values, and then we use Equation 3 to update these values. This update process is repeated until convergence.

$$V_{t+1}(s) = R(s) + \gamma \max_{a} \sum_{s' \in S} T(s, a, s') V_t(s'), \tag{3}$$

where $R(s)$ is the immediate reward, and a is the taken action.

2) POLICY ITERATION

This method differs from the value iteration in what gets updated. Instead of updating the value functions, we update the policy. It can be summarized in three steps:

1) Initialize π with random values.
2) Use π to calculate the value function V.
3) Change π iteratively until convergence using Equation 4.

$$\pi_{t+1} = \underset{a}{\operatorname{argmax}} \sum T(s, a, s')V_t(s') \tag{4}$$

3) MONTE CARLO METHOD

The Monte Carlo method is a model-free RL approach where only a record of states, actions, and rewards resulting from the agent interactions with the environment is used for learning [23]. The task is divided into episodes, where each episode ends with a terminal state, and then after reaching this end, the episode data is fed back into the model to improve the policy. Firstly, we follow the old policy to generate a list of episodes. Then, we iterate over all these episodes and update the value function for each episode by using its average returns from the current episode, and all previous episodes in which this state was seen the first time [23].

4) TEMPORAL DIFFERENCE

Temporal Difference (TD) is a model-free RL approach that combines between dynamic programming (includes policy iteration and value iteration) and Monte Carlo method [23]. It is like Monte Carlo method in the way that no transition function is required, and at the same time, we don't need to wait until the end of the episode in order to update the parameters (as in dynamic programming method) [23].

5) $Q-$LEARNING

It is a model-free RL method to give an agent the ability to act optimally in a MDP or POMDP setup. $Q-$learning objective is to learn a policy that tells the agent what action to take at which state by maximizing the total reward. The total reward consists of the immediate reward and the expected value of the future rewards penalized by the discount factor γ. The optimization is done by having a function that calculates the **Quality** of every decision (taking action a at state s). This function can be written as:

$$Q : S \times A \rightarrow \mathbb{R} \tag{5}$$

At the start, Q is initialized with random values (or by using another initialization mechanism). Then the agent start learning as follows:

Starting from state s_t at time t, the agent takes action a_t based on the current Q function. It receives a reward r_t and its state changes to s_{t+1}. Based on this reward, the Q function is updated using the following equation:

$$Q^{new}(s_t, a_t) = (1 - \gamma) \cdot Q^{old}(s_t, a_t) + \alpha(r_t + \gamma \cdot \underset{a}{\operatorname{argmax}}\, Q(s_t, a)) \tag{6}$$

where α is the discount factor, r is the reward and γ is the learning rate.

6) FUNCTION APPROXIMATORS

$Q-$learning creates a table with the Q values of all possible combinations of states and actions. This can be infeasible when the state space is very large. To solve this problem, several methods are used to replace $Q-$table by approximating it using a function that maps state-action pairs to the $Q-$values [23]. Different methods such as linear regression, decision trees [25], and artificial neural networks can be used as function approximators.

V. HIGH LEVEL OVERVIEW

In order to help the reader grasp the full picture of the current RL interpretation methods, we provided a high level classification of these methods as illustrated in Figure 1. We divided the methods into five main groups: decision trees, summarization, computer vision, natural languages, and custom models. Then at each parent node we connected its subgroups. At the leaf nodes of the graph we added the papers that belong to each category.

VI. CATEGORIZATION METRICS OF RL INTERPRETATION METHODS

RL interpretation methods -similar to ML's- can be categorized into groups based on different metrics [8]. In this section, we first defined different categorization metrics, and then we applied them on the interpretation methods shown in Figure 1 and described in more details in Section VII.

A. EXPLANATION TYPE

RL interpretation can be performed in three different ways [26]: using another method to generate explanations while keeping the original model (Post-Hoc), introducing a new interpretable learning model (Intrinsic), or a combination both methods by altering the original model to improve the interpretability instead of replacing it entirely. The following sections show this classification applied on the methods described in Section VII.

1) POST-HOC METHODS

These methods are used beside the original model to help users understand the decisions and details of the model. However, the model architecture is left as it is. All summarization methods explained in Section VII-C belong to this category. Moreover, all the computer vision explanations described in Section VII-B (with the exception of intrinsic saliency maps explained in Section VII-B1b) are post-hoc methods. Natural language explanations in Section VII-E are also classified under this category. Finally, the model reconciliation method explained in Section VII-F belong to this category too, since the robot model is left as it is which represents the original black-box model.

TABLE 1. Basic classification of RL interpretation methods.

	Explanation Type			
	Post-Hoc		Intrinsic	
	Local	Global	Local	global
Textual	Template-based, Query-based			
Images	Post-hoc Saliency, Counterfactuals		Intrinsic-Saliency, RS-Rainbow	
Rules		Reconstruction, Interstingness		Decision-Trees, Custom Models
List				State Importance

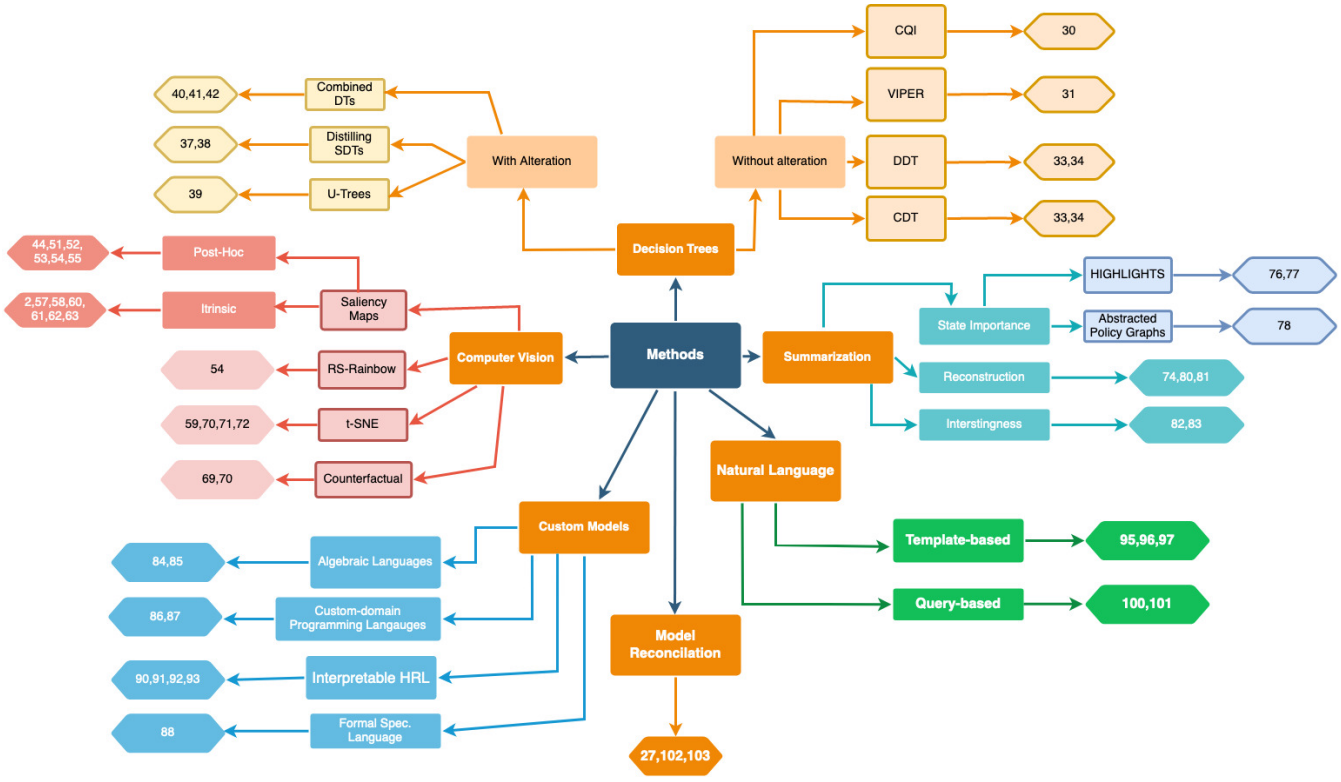


FIGURE 1. General overview of RL interpretation methods.

2) INTRINSIC METHODS

Intrinsic methods relies on using inherently interpretable models such as logistic regression and decision trees [8]. Since tree-based methods explained in Section VII-A replace the original black-box model by some sort of decision trees, they belong to this category. In addition, we also think of custom learning models described in Section VII-D as intrinsic interpretation methods since the original black-box model is replaced.

3) COMBINATION OF BOTH

Sometimes the actual model is altered in some way in order to give explanations instead of replacing it entirely. This was applied on attention based saliency maps that we explained in Section VII-B1b, where the basic learning model is used with more layers to produce the saliency maps.

B. SCOPE OF INTERPRETATION

RL interpretation methods can either explain:

1) **Local** decisions of the model such as why a specific action or group of actions were chosen by the agent.
2) Its **global** behavior or strategy in taking decisions.

This metric was inspired by Molnar [8] in their interpretable ML book where they applied this concept on ML interpretation methods. We should also note that this metric is related to model replacement concept explained in section VI-A2. Because replacing a black-box model with a new interpretable learning model is done for the whole model and not only a subset of samples, all intrinsic methods described in section VI-A2 are considered as global interpretation methods. However, the opposite is not true. Post-hoc interpretation methods can be designed to get insights about global behavior, or to give interpretations for a specific decision. As a result, decision tree based methods explained in section VII-A and custom learning models described in section VII-D are global. In the following sections, we comment on the scope of the other RL interpretation methods.

1) GLOBAL METHODS

The following RL interpretation methods are global:

1) HIGHLIGHTS method in section VII-C1a because it generates summaries to explain the agent policy.
2) Abstracted Policy Graphs (APG) method explained in section VII-C1b since the generated graphs represent the policy.
3) Model reconstruction method in section VII-C2 is used to generate a list of state-action pairs in order to explain the agent's behavior generally and not based on a single decision, and hence it is a global explanation method.
4) Interestingness summaries in Section VII-C3 are generated to explain the policy, and hence, classified under global interpretation category too.
5) t-SNE method explained in Section VII-B4 is global, since it abstracts the whole state space.

2) LOCAL METHODS

The following RL interpretation methods are local:

1) Saliency maps in Section VII-B1 is used to explain the agent's local decisions such as why a specific move in a game is chosen and hence are local models.
2) Counterfactual method in Section VII-B3 as every counterfactual image is generated for a specific sample.
3) Query based explanations in Section VII-E2, since they are used to answer a **specific query**.

C. EXPLANATION FORMAT

The output of the interpretation method (explanation) can be produced in different formats that can be classified as follows:

1) **Textual Explanations:** The explanation is formatted in natural language. All methods under Section VII-E [27] belong to this category. They are actually classified based on this property. In addition, the model reconciliation method described in Section VII-F is also textual, since the model differences are shown to the user in a natural language format.
2) **Image Explanations:** In these methods, explanation is shown as an image. This category includes the methods described in Section VII-B.
3) **List of States or State-Action Pairs:** Here a list of states or pairs of states and actions are chosen based on some criteria to produce the final explanation. The summarization methods explained in sections VII-C1 belong to this category.
4) **Rules:** Here the explanation is shown as rules that humans can apply to understand the final decision of the agent. We consider all types of decision trees as rules in addition to the formats. As a result, methods under Section VII-A and Section VII-D2 belong here.

D. ORDER OF APPLYING THE MODEL

The interpretation can be applied before training the model, during the training, or after. The order of applying the interpretation method can produce different insights. Pre-model interpretation gives a better idea about the environment setup. On the other hand, applying interpretation methods during the training helps us to understand the learning procedure, and can clarify issues such as whether the agent is actually learning, doing random weight changes, or memorizing [26]. Post-training interpretation methods explain what the model actually learned. The order of applying the method is affected by whether the interpretation method is an intrinsic or post-hoc VI-A. For example, in intrinsic methods, since we have a new learning model that replaces the black-box model, the order of applying the interpretation is irrelevant. As a result, we only consider this metric for post-hoc interpretation methods.

1) ORDER OF POST-HOC METHODS

In this section, we comment on the order of applying the interpretation on post-hoc interpretation methods as follows:

a: SUMMARIZATION METHODS

For summarization methods, we can categorize the order of applying the interpretation into two groups;

1) State importance and model's reconstruction methods explained in sections VII-C1,VII-C2 are applied after training the model.
2) Interestingness method in Section VII-C3 has three components that are applied during the three phases.

b: COMPUTER VISION

In computer vision interpretation methods explained in Section VII-B, both the saliency map and gaze method are applied after the training.

c: NATURAL LANGUAGE

All natural language methods explained in Section VII-E are applied after the training.

E. MODEL AGNOSTIC VS MODEL SPECIFIC

Interpretation of a complex model can either be performed by replacing the complex model with an inherently interpretable model such as decision trees, or by utilizing another method beside the complex model to generate the explanations. The first type is called *intrinsic*, while the second type is called **post-hoc** [8].

Decision trees methods explained in Section VII-A and the custom models explained in Section VII-D are both intrinsic interpretation methods. On the other hand, natural languages-based approaches and summarization methods are post-hoc. For computer vision methods, we have both types. For example, saliency maps can be either intrinsic or post-hoc based on the way we implement them as shown in Section VII-B1. The other sub-groups of computer

TABLE 2. High level overview of tree-based methods.

Method	Applications	Nodes #	Replacement?
VIPER	Cart-Pole, Pong	<1K	Explanation
CQI	2D navigation	<100	Replacement
DDT	StarCraft	N/A	Replacement
Distilling SDTs	Mario game	depth <8	Explanation
U-Trees	Cart-Pole, Mountain-Car	100 - 10K	Explanation
MoET	Cart-Pole, Pong, Acrobat, Mountain-Car	?	Explanation
Gradient Boosting	Cart-Pole, Mountain-Car	?	Both

vision (namely: RS-rainbow, t-SNE, and counterfactuals) are post-hoc.

F. FIDELITY & ACCURACY

Fidelity means the extent to which the interpretation is truthful to the actual model [8], [28]. Before using the interpretation, we need to make sure that it explains the actual model, and hence fidelity metric is important [8]. Measuring fidelity is affected by the scope of interpretation methods. For example, local interpretation methods are truthful to their corresponding black-box model only on the limited region of inputs they intended to explain [8]. Papenmeier *et al.* [29] studied the effect of explanations fidelity on gaining users' trust. They found that while the accuracy of the model has a higher effect, low-fidelity explanations can decrease user's trust.

Fidelity is more important for post-hoc interpretation methods since they are used to explain the black-box model instead of replacing it entirely. On the other hand, model accuracy is more important for intrinsic methods because after replacing the black-box model by an inherently interpretable model, the alignment between the two models is not as important as getting a high accuracy from the new interpretable model, since we are not going to use the old black-box model anymore. Generally, measuring fidelity and comparing between different methods is not an easy task, we will comment on it whenever we have data inside while explaining each method in Section VII.

In summary, Table 1 shows the application of three categorization metrics on the RL interpretation methods at the same time (The interpretation methods are explained in more depth in Section VII). The three metrics included in the summary are: explanation type, scope of explanations, and explanation format.

VII. RL INTERPRETATION METHODS

This section states and classifies different methods used to interpret RL.

A. USING DECISION TREES

Since decision trees can be visualized textually or graphically very easily (inherently interpretable) [30], they are used in several formats to replace the original black box model for the purpose of interpretation. However, learning a decision tree is not an easy process [31]. As a result, in several RL decision tree-based methods, different tricks in the learning algorithm are used, such as imitation learning. Imitation learning or distillation is guiding a shallow model such as decision trees by a more complex model like DNNs [32]. In other words, instead of training the shallow model on the dataset directly, the DNN is trained first on the data, and then the trained DNN transfer its knowledge to the shallow model by generating a soft-labeled dataset. A soft-labeled dataset is a dataset with continuous values for the labels instead of the original hard-labeled datasets. The complex model in imitation learning setup is called *the teacher* while the shallow one is called *the student*. Before explaining these methods in detail, we first give a high level comparison between them based on the applications that the authors tested their methods on, the average number of nodes in the generated trees, and whether the method is originally used to replace a complex model or just to explain its decisions. This comparison is given in Tab. 2. Note that we can not compare the number of nodes directly since we do not have a unified application. For example, it is not possible to compare CQI to VIPER (see sections VII-A1b and VII-A1a) because CQI was tested on a simpler problem, but at least we can get an idea about the length of the trees that each method is able to generate. Although some games such as Cart-Pole were tested for most of the methods, we need to set up a unified experiment that includes all the methods in order to come up with a more solid comparison.

1) DECISION TREES WITHOUT ALTERATIONS

a: VERIFIABILITY VIA ITERATIVE POLICY ExtRaction (VIPER)

Bastani *et al.* [31] replaced RL black-box models with decision trees. They also proposed their own learning algorithm that applies imitation learning from a DNN model. The authors used the $Q-$function of the DNN model (the teacher) to generate values for training the decision tree model (the student), in addition to the optimal actions for each state (the learnt policy). This imitation learning-based method is called VIPER. It was applied on two different Atari games: Cart-Pole and Pong. Although the main focus of this method is verification, it can be considered as an RL interpretation method, since decision trees are inherently interpretable. However, the relatively high number of nodes in their final output (limited by a maximum of 1000 nodes) makes it harder for humans to comprehend the output. Moreover, we reached maximum accuracy within this limit only on simpler versions of Pong and Cart-Pole, which open questions for the applicability of the method on more complex problems. The authors applied their method for model-based RL policies.

b: CONSERVATIVE Q-IMPROVEMENT (CQI)

CQI [30] also interprets RL models by using decision trees. In contrast to the previous method, CQI can be applied for model-free RL such as $Q-$learning which was the main focus of the authors. The main contribution of this method is its ability to generate shorter trees (less than 100 nodes for a toy navigation problem), since adding new nodes to the final tree is constrained by the amount of the discounted future reward we gain from the additions. As a result, we need to set a threshold that reflects the trade-off between the length of the tree (and subsequently interpretability), and the accuracy of the RL policy in performing the task at hand. The resulting decision tree consists of two types of nodes: branch nodes and leaf nodes. Each branch node has two children, and the split is based on one of the features of the state vector, while a branch node represents the action that the agent takes for the path ends with this node. Because of this, CQI can be applied only when it is possible to represent the state with a features vector. The authors implemented their own learning algorithm and no other external complex model is used for training as in VIPER. Similar to VIPER, this algorithm was applied on a simple 2D robot navigation problem, where the agent has to move towards the goal and avoid the obstacles.

c: DIFFERENTIAL DECISION TREES (DDT)

Rodriguez *et al.* [33], [34] proposed a method that updates the tree entirely during the learning phase instead of adding the nodes incrementally as in the previous two methods. They made this possible by introducing a differential decision tree so that gradient decent algorithm can be applied. The main difference between DDT and normal decision trees is using sigmoid function for splitting the nodes instead of the non-differentiable if/else rules. The leaf nodes in the resultant tree represent single features because the authors applied a discretization technique by replacing the features vector with the maximum feature. This algorithm was applied on three Atari games: Cart-Pole, Lunar Lander and Acrobat. According to the authors, the accuracy of their model outperformed their MLP model which was mainly created for comparison purposes. However, the authors did not comment on the length of the final decision tree, despite the fact that it affects the interpretability, since it is hard for humans to comprehend decision trees with large number of nodes. It is worth mentioning that the authors were able to approximate a policy for StarCraft II Micro-battle using a tree of only 15 nodes. This method was applied on $Q-$learning RL.

d: CAUSAL DECISION TREES (CDT)

Madumal *et al.* [35] used decision trees to show causal explanations. To implement this, first a decision tree is trained using a list of state-action pairs. The authors limited the depth of the decision tree to be as large as the size of the possible actions (size of action set). Similar to other decision trees-based models such as CQI, the leaf nodes in the decision tree represent the actions while the split (spilt nodes) is based on the state features. Then an action influence graph is used to generate casual explanations from the fitted decision tree nodes. The action influence graph is constructed by using the influence of actions on variables (features of the state) [36]. The explanations resulting from this method are not just for justifying why a specific action was taken, but also used to explain why the agent did not take an alternative scenario. The authors conducted an experiment to evaluate their method by asking humans to predict the agent decisions based on the given explanation, and then the accuracy of this prediction is used to evaluate the quality of the generated explanations.

2) ALTERED DECISION TREES

In these methods decision trees are used with adding some alterations such as using probabilities inside nodes or adding linear equations to leaf nodes.

a: DISTILLING SOFT DECISION TREES (SDTs)

Coppens *et al.* [37] proposed a method for interpreting DNN RL models using SDTs. They used imitation learning to transfer the knowledge of the DNN model into the SDTs. SDTs method is a combination between binary decision trees and neural networks. In SDTs, branching nodes represent individual perceptrons and the output the probability p of moving to the right sub-tree (and accordingly $1 - p$ is the probability of moving to the left sub-tree), while the output nodes represent the actions. This method was applied on Mario AI benchmark [38] where a DNN agent was trained to play Mario game. The application of this method is more complex compared to the previous tree-based RL methods mentioned above in terms of both the state, complexity of the environment, and the number of the allowed actions. The authors trained a DNN of type actor-critic, and then its information was distilled to the soft decision trees. They performed the imitation learning by generating pairs of states and actions from the trained model, and then they fit SDTs of different depths ranging from 3 to 7. Note that each node in the tree includes a complete set of weights equal to the number of features. For humans to understand the decisions of this model, they have to traverse the tree from the top node to the leaf nodes. This process gets harder with increasing depth. The authors found that the fidelity of the model decreases with increasing depth and they have to sacrifice fidelity if they want to generate more understandable explanations. However, it is not fair to compare the fidelity of this approach with the previous decision tree based methods since the application (Mario game) is more complex.

b: LINEAR MODEL U-TREES

Liu *et al.* [39] also used imitation learning to distill the knowledge of a DNN RL model into their proposed decision trees-based model which is called *linear model U-trees*. The authors tried two different methods to generate the data from the DNN model: saving all state-action pairs with their

corresponding $Q-$values during training, and generating data online from the trained model by making it interact with the environment. The state is represented in this method by a vector of features, while branching nodes are used to split the tree using if/else rules based on one of the features just like normal decision trees. They differ mainly in having a different linear model in each leaf node that covers all the features, which is responsible for producing $Q-$values for its corresponding path. The authors used Stochastic Gradient Descent (SGD) to train the linear models in the leaf nodes and the splitting branching nodes are added when the improvement (in rewards) introduced by the linear layer is not enough. This method was tested on three games: Mountain Car, Cart-Pole, and Flappy Bird. The state space was unified to be an 80*80 grey-scale image. The authors compared their method with two different methods in building general decision trees (not used explicitly for interpretability). They concluded that u-trees is better than the baseline methods in both fidelity and the number of nodes used. However, the number of nodes is still very large and hard to visualize (hundreds to thousands of nodes). One of the possible reasons might be the complexity of the application.

c: COMBINING SEVERAL DECISION TREES

Sometimes several smaller decision trees are combined in different formats as follows:

1) **Mixer of Expert Trees (MoET):** The main idea of MoET [40] is to use multiple smaller and specialized decision trees to interpret a DNN-based RL policy. The authors adopted the same learning algorithm used in VIPER with replacing the decision trees in VIPER (the student) with the mixture of expert trees. Generally, Mixture of Experts (ME) is a model that consists of several models (experts) where each one specializes in a sub-space of the inputs. The specialization is performed by using a gate function that determines the contribution of each expert in a specific input sample [41]. To get the final result, a weighted sum of the experts is calculated.
   Vasic *et al.* [40] used decision trees as expert models and combined them by a soft-max gating function. The authors followed the same imitation learning approach of VIPER to transfer the knowledge from the RL DNN model to MoET. Instead of using all experts with different probabilities as in general ME models, in MoET only one expert is selected for the explanation to make it more understandable. This way of selection might be easier for a human to understand, but it decreases the fidelity of the given explanation. The authors tested their method on Cart-Pole, Pong, Acrobat and Mountaincar Atari games and compared their results with VIPER. The authors found that their approach is better than VIPER in both fidelity (the alignment between the teacher and the student) and performance (as expressed by the rewards).
2) Gradient Boosting Decision Trees: Similar to the MoET method mentioned above, gradient boosting is a different way of combining a group of models together to come up with the final decision. In this group of models, the first one is trained on the whole data and the second model is trained on the portion of data where the first model made errors and so on. In other words, each model corrects the mistakes of its predecessors. The resulting individual models are combined together in a weighted sum format to give the desired output for a given input. Brown and Petrik [42] used this approach to interpret RL models using decision trees. Moreover, they also proposed retraining the recent trees with the knowledge of the older ones in order to solve the problem of having large number of trees.
   This method was tested on two Atari games: Cart-Pole and Mountain-Car. It is worth mentioning that the authors also used imitation learning by training a complex model first (neural network for Cart-Pole and SARSA model for Mountain-Car), and then they used it for generating data for their model. Although one of the critical motivations behind using this method is interpretability, the authors did not provide a comparison for these methods in terms of some interpretability measures such as fidelity and the number of nodes, and focused instead on the performance of their method in terms of the total reward they were able to collect.

B. COMPUTER VISION EXPLANATIONS

In this category we list the methods where the explanation is shown as an image, a list of images, or an image with some other corresponding information.

1) SALIENCY MAPS

A saliency map is an image that highlights the contribution of individual pixels to the model's decision. For example, pixels with higher value in the output image may indicate higher importance for their corresponding locations in the input image. As a result, changing the values of these pixels in the input image should affect the output more significantly [2]. The generated image can be either 1) local to explain the model decision for a specific image, or 2) global to generate the important pixels for a chosen class across the model as a whole. The latter is used to justify why that class is picked instead of the others. Saliency maps interpretation method was used originally to explain ML models such as DNNs [43] and ConvNets [44]. For example, Simonyan *et al.* [44] introduced a method that uses the gradient of the output with respect to the input image to generate both local and global saliency maps.

Generally, there are different methods to generate saliency as reviewed by Nikulin *at el.* [2]. They investigated these methods and categorized them into post-hoc and built-in saliency maps. In post-hoc methods saliency maps are generated after training the model, while in intrinsic approach the same learning model is used to generate the maps as well as

the class predictions. We should note that this categorization metric is similar to explanation type that we described in Section VI-A. However, we used *intrinsic* term borrowed from the authors of [26] instead of *built-in* with the same meaning.

Post-hoc methods were used broadly in different formats to explain ML [2]. For instance:

1) **Guided BackProp** Springenberg *et al.* [45] proposed this method to test the importance of pooling layers in ConvNets. Their method is based on deconvolution, where the direction of the ConvNet is reversed to produce an image from the high level feature maps. For each neuron that got activated, its corresponding pixels are highlighted in the resulting image. This is performed by using the gradient of the neurons conditioned on the input image. Shrikumar *et al.* [43] introduced DeepLift where they used back-propagation to calculate the contribution of every pixel to the final prediction. For any input image, a reference input is selected based on the application domain. Then the difference between the predicted class and the opposite class is expressed in terms of a difference between the input and this reference.
2) **Layerwise Relevance Propagation (LRP) [46]:** It generates the contribution of every pixel to the final prediction by decomposing the classifier first into multiple layers. This decomposition is done pixel-wise which results in scores for single pixels.
3) **Integrated Gradients:** One of the problems of the previous two methods (DeepLift and LRP) is that sometimes they generate different saliency maps for equivalent neural networks. We call two neural networks equivalent if they produce the same outputs when the inputs are identical despite their differences in implementation. Equivalent networks should generate the same saliency maps and this property is called implementation invariance. Sundararajan *et al.* [47] proposed Integrated Gradients approach which preserves implementation invariance by using continous gradient instead of the discrete gradients used in the other two methods.
4) **SmoothGrad [48]:** To generate a saliency for a specific input image, difference copies of it are created with adding noise, and then a saliency map is generated for each one. After that, these saliency maps are averaged to get the final representative output. This process is aimed to remove the noise from the resultant saliency. Similarly, VarGrad [49] method follows the same methodology of SmoothGrad, and the only difference is using the variance instead of the mean. Seo *et al.* [50] found that SmomothGrad conflicts one of its basic objectives by not making the gradients smooth, while VarGrad was independent of the Gradient of the scores.

In the following sections we describe different saliency maps methods used to interpret RL:

a: POST-HOC SALIENCY MAPS

Post-hoc saliency maps can be divided into the following:

1) **Backpropagation-based Saliency Maps:** This approach is based on the work of Simonyan *et al.* [44]. It produces two types of saliency maps: one to show the importance of image locations for the whole class, and another one for just one input image. They used back-propagation to compute the derivative of the output with respect to every input pixel and then this derivative is converted to the image dimensions to be visualized. The conversion depends on whether the original image is grey-scale or RGB. For grey-scale images, the absolute values of the derivative are taken directly, while for RGB images the maximum channel value is chosen for each location. Simonyan *et al.* [44] method was utilized by Wang *at el.* [51] to interpret their dueling Q-network that has two streams: one to estimate state values and the other to estimate action advantages. The two streams are then aggregated to produce the Q values for the state-action pairs. The authors used Simonyan *at el.* [44] method to generate maps for the two streams separately. When they applied this on a car Atari game, they found that saliency maps for the state values pay attention to the road while the saliency maps for the action scores pay attention only when there are cars to avoid collision. Their application proved that this method helps to understand the behavior of the two networks separately.
2) **Perturbation Saliency Maps:** Unlike backpropagation method, perturbation is used (different changes to inputs) instead of taking the derivative. The reason behind that is sometimes the generated saliency from backpropagation-based methods does not make sense from a practical point of view, because it explains the prediction by changes to meaningless pixels [52]. Perturbation methods solve this problem by making custom meaningful changes to the input images that affect the final prediction more significantly. Moreover, they are also more model-agnostic since no constraints such as being differentiable are required from the model [53]. On the other hand, perturbation methods suffer from the need of making multiple passes in order to generate a saliency for a single input image [54]. Greydanus *et al.* [52] proposed their own perturbation algorithm of generating saliency maps and then applied it to understand the agent's behavior in different Atari games to perform the following goals: visualizing the agent's policy during learning phase, following the reasons behind badly-performed agents, and identifying wrong situations of giving the agents high rewards.
Perturbation methods may generate saliency maps that do not highlight only the relevant features to the action we need to explain, but the other actions too. Puri *et al.* [55] proposed Specific and Relevant Feature Attribution (SARFA) method that avoids this problem by keeping the perturbations that alters only the action

we need explain and ignoring changes to the expected rewards applied by other actions.

b: INTRINSIC SALIENCY MAPS

Instead of using another model to generate saliency maps, in this interpretation method the original black-box model is altered to produce the maps in addition to the predictions. Altering the model to generate saliency maps beside its main objective most probably results in having a lower performance, which makes it harder to keep both interpretability and accuracy [2].

To the best of our knowledge, all these methods utilize some sort of attention mechanism. Attention can be described as adding a new layer (vector of weights) to the DNN to represent how much attention is payed for each input pixel (in case of visual inputs) [56]. One example of these methods is Deep Attention Recurrent Q-Network (DARQN) [57]. It is an extension to Deep Recurrent Q-Network (DRQN) proposed by [58]. In DRQN, an LSTM is used in the architecture of DQN [59] in order to represent temporal patterns more efficiently. Sorokin *et al.* [57] altered DRQN by adding attention mechanism to it. They built an architecture that consists of a convolutional, attention, and recurrent networks. The feature maps produced by the convolutional layer are applied to the attention layer that produces a list of vectors with the same number of feature maps and each vector has the same dimensions of the input image. The authors applied two types of attention mechanism: soft attention where a weighted sum of the vectors is calculated to produce the saliency map, and hard attention where at every time step only one pixel (attention location) is selected. DARQN was applied on several Atari games and the results were comparable to DQN and DRQN. The interpretation here is provided by the saliency maps generated from the attention layer. Similarly, Mousavi *et al.* [60] used the same architecture with soft attention mechanism to play different Atari games. They also used two metrics to compare the ability of the generated maps to identify the fixation locations across different models: Normalized Scanoath Saliency (NSS) and Area Under Curve (AUC). The two metrics compare the saliency map and the locations that humans focus on the most. This human data is obtained by asking people about the places they consider important while playing the game. In contrast, Zhang *et al.* [61] used a different approach for collecting human data. They used eye tracking sensors to collect the locations of human's gazes while they are playing. Then this collected data are used to train a network that predicts human gazes (both position and saliency maps by using attention). After that they built a DNN that predicts the actions. An interesting aspect about this approach is that attention is learnt from humans by utilizing the eye tracking data and instead of just utilizing it for evaluation like the previous method [60]. Nikuli *et al.* [2] suggested a quantitative method to evaluate the generated saliency maps by comparing them with the eye tracking data publicly available by [62]. They used their proposed metric to compare different attention based RL models. In addition, Querishi *et al.* [63] proposed a similar architecture that differs in using recurrent attention models (RAM) [57] for the attention layer. They also used a dual network where a two distinct networks for the value function and state scores are created. This architecture is inspired by [51] as described in backpropagation-based saliency maps above. Querishi *et al.* [63] applied their architecture on a human-robot interaction problem and used the attention in order to show people robot attentions when interacting with it.

c: CHALLENGES OF SALIENCY MAPS

Saliency maps also have some problems and challenges that need to be solved. Adebayo *et al.* [64] proposed an experiment that can be used to compare between the performance of different saliency maps methods. It consists of basically two tests: a model parameter randomization test, and a data randomization test. The first test assures whether a randomly and untrained network with the same architecture can produce a similar result to the model under testing, while in the second test we first change the data labels randomly, and then retrain the model to see the effects of this new altered dataset on the resulting saliency maps. Within the four methods tested by the authors, two passed their tests (Gradients and GradCAM) while the others failed (Guided GradCAM and Guided backProp). Note that this test was conducted on a limited set saliency maps generation methods. Moreover, it does not take the sequential nature of the RL systems into account [65].

In addition to the subjectivity problem, and according to a survey performed by Atrey *et al.* [65], saliency maps in RL interpretability suffer from unfalsiability and cognitive bias. In order to face these challenges, the authors created a methodology to test the conclusions drawn from the generated saliency maps. They applied this by formatting the conclusions in the following format X = "*{concept} is salient* $\longrightarrow$ *agent has learned {representation} resulting in {behaviour}*". For example, we can fill this template by the following data from an autonomous driving agent: stop signals text is salient $\longrightarrow$ agent has learned to identify stop signals resulting in taking a complete stop. The concept is the body represented by the pixels in the saliency, representation is the abstract concept like "identifying the enemy in a game", while the behaviour is the agent action or sequence of actions. After doing this, the state is reversed, and a counterfactual saliency map is generated to test whether this pattern given in the format above holds. If it turned out to be the case, then we have a stronger evidence that the agent has actually learnt the concept stated in X. The authors concluded from their study that saliency maps should be used for exploration rather than explanation since it is not easy to draw a solid conclusion from them [65]. The authors suggested using saliency maps with other supporting tools, and we think this experiment can be integrated with the two tests proposed by Adebayo *et al.* [64] as explained above to get a clearer picture about the behaviour of an RL agent.

Furthermore, RL systems have two properties that make explaining them by saliency maps a harder [66]; the non-deterministic nature of the policy, and the non-static (temporal) pattern of the inputs. It is not easy to capture these two properties by saliency maps.

2) REGION SENSITIVE RAINBOW (RS-RAINBOW) METHOD

Yang *et al.* invented Region Sensitive Rainbow (RS-rainbow) interpretation method [54] to identify the important regions in the input image (rainbows) while taking actions in deep $Q-$learning setup. The difference between saliency maps and this method is that here the output is a part of the original image while the pixels of saliency maps represent the importance (e.g., in terms of intensity). Rainbows are generated by an architecture similar to attention method used to generate built-in saliency maps. The authors incorporated different techniques in RL such as double DQN [67], Dueling DQN [51] and multi-step learning [68] in their architecture [54] which consists of three main components: **image encoder** that transforms the input frames into feature maps, **region-sensitive** module that outputs the local image probabilities used to generate the important regions and **policy layers** to output $Q-$values. In addition to providing built-in interpretation, this method also outperforms DQN method as it has a more efficient representation for the states since it incorporates the most important region as a part of this representation.

3) COUNTERFACTUAL STATES

A counterfactual state is a state which is slightly different from the original state, but it leads to a different action. Dhurandhar *et al.* [69] followed this approach to explain ML models by highlighting both the important pixels in the given output (their presence is important), as well as the one which their absence is critical in producing the output class. Generally, the goal of this method is similar to saliency maps method explained in section VII-B1: trying to identify which aspects of the visual input are important in agent's decisions. However, instead of getting an image that shows which pixels are more important in choosing actions, a new image with the minimum changes is created instead. Olson *et al.* [70] developed a deep generative network model to produce counterfactual examples that can be added to an already learned agent (the underlying learning model need not to be altered). In order to do this, first the input image is converted into a latent space, and then in the next phase this space is converted to actions. The authors proposed a deep network architecture that consists of a generator, discriminator and an encoder. The model is trained by a list of state-action pairs generated from the learned model. The authors added an adversarial autoencoder in order to make the generated counterfactual more realistic. By realistic we mean an image that can occur in the real world. For example, it is not realistic to have an image with two players in a one-player Atari game. Furthermore, the authors wanted to make humans involved in evaluating the explanations, and conducted an experiment by asking non-experts to evaluate whether the generated counterfactuals are realistic. They found that, on average, their results are not far from being realistic. Generally, one of the challenges of counterfactual states is how to define the minimum change between the generated state and the original one [69]. The difference that was defined by equation to be minimal might not be expressive for humans. Moreover, we might have more than one counterfactual example with the same difference from the original state (within some threshold) and it is a new challenge to determine the best one since the evaluation contains a subjective part.

4) t-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING (t-SNE)

t-SNE is a dimensionality reduction technique used for data visualization by converting a high dimensional data into two or three dimensions [71]. This method keeps the similar points (in terms of distance) close to each other in the new dimensions.

Zrihem *et al.* [72] proposed a method that utilizes t-SNE for RL policy explanation. Their method first deduces a semi-MDP space from a DQN RL that consists of a sequence of states with their activations and Q values. Then t-SNE is applied on this space and the generated t-SNE points are clustered. The next step is to evaluate the transition function for these clusters by utilizing the temporal patterns in the previously generated sequences. Finally, the clustered t-SNE points are converted into images from the original inputs by calculating the average image of each cluster, and then these images with their corresponding transition probabilities are visualized in a directed graph. This method is useful in summarizing the transition function of a model-free RL system such as DQN. However, it lacks a proper way of evaluating the results, and hence, some good tests similar to these of saliency maps explained in Section VII-B1.c should be developed.

Zahavy *et al.* [73] improved this approach by generating saliency maps and adding them to the visualization beside the original images. Mnih *et al.* [59] also used t-SNE method to visualize the state representation of a DQN agent for Space Invaders Atari game. They found that the t-SNE method tends to group the embeddings of the perceptually different images together if they have a similar cumulative reward.

C. SUMMARIZATION METHODS

A summary is generated to describe the agent's interaction with the environment in different formats. Some summarization methods aim to describe the agent's policy in a limited set of states while others try to state the minimum possible information enough to reconstruct the agent's model by humans [74]. Amir *et al.* [75] studied different methods in literature and proposed a conceptual framework for building policy summarization systems. They suggested three important conditions that should exist in any summarization method:

1) It should state the important states from the human's prospective.
2) It should add a way to transform state space into a new more understandable format.
3) Human input is important in determining the format of the summary output.

These stated conditions might be useful in evaluating the quality of the generated summary. However, since they include the human factor, it is hard to quantify their results. We should also note that state importance is not the only way of RL policy summarization, and there are different methods as we will see in the next sections.

We can classify the summarization methods based on their explanation format. It might be a subset of the agent states either in their original format (state importance), or in a new dimension (abstracted policy graphs). In addition, they can be in a format of a scattered list of information such as interestingness method, or a set of the minimum information that can be used to reconstruct the RL model by humans. Based on this, we can classify the current summarization methods into the following:

1) STATE IMPORTANCE

Here the summary is generated based on a limited set of states that are indicated to be important. The definition of this importance is application dependent. We can list the following methods under this category:

a: HIGHLIGHTS

HIGHLIGHTS method proposed by Amir *et al.* [76] is used to explain model-based RL policies in a list of trajectories. Each trajectory consists of a sequence of important state-action pairs. According to the authors, a state is said to be important if taking different actions from that state leads to different scenarios (e.g., taking an exit in a high way VS not taking it). The number of the trajectories is a configurable parameter and it depends on how much time the users can spend in watching the summary before losing attention. The same applies on the number of pairs within each trajectory. The authors conducted an experiment that involves humans to see whether their explanations help users to evaluate the agent performance more efficiently. They provide the users with the summaries of several agents with different performance, and asked them to choose the best performance based on the shown explanation. The authors found that HIGHLIGHTS improved the capability of humans in choosing the most performing agent. However, sometimes the important states repeat in multiple trajectories which decreases the richness of the explanation. In order to solve this, they propose another variant of their approach called HIGHLIGHTS-DIV, which has another constraint to prevent repetition. The authors found that HIGHLIGHTS-DIV is better than the basic version of the algorithm in helping users choosing the best performing agent. Similarly, Huang *et al.* [77] proposed a method that helps human users to build trust by showing the most critical states, and then evaluating the decisions taken at theses states by humans. This method is useful for applications that require robot-human interaction, because it can be used to decide when the humans should take control.

b: ABSTRACTED POLICY GRAPHS(APG)

APG method proposed by Topin and Veloso [78] aims to explain the agent's local decisions by first creating a connected abstract state space. The method is applied only on model-based RL systems (formulated by MDP), since the transition function is needed to generate the abstract space. The abstracted states are created by grouping the states based on the Feature Importance Ranking Measure introduced by Zien *et al.* [79]. This measure takes a set of states and returns the importance of each feature in these states so that if the feature tends to take the same values it is pointed as less important. This method is similar to the t-SNE method proposed by [72] in the way that it maps the states into another abstracted space and visualizes them in a directed graph with probabilities. However, APG needs the transition function and can not be applied for model-free methods as t-SNE. In addition, the method of mapping to the law dimensional space is different (features importance compared with t-SNE).

2) MODEL RECONSTRUCTION

The philosophy of building summaries is different in this approach. Instead of applying a method to extract the important states or state-action pairs, we assume that humans have some implicit models for the RL agent, and then we try to give them the minimum possible amount of information enough to make their model more representative to the actual agent model. In other words, we think of a summary as a medium that can be used by humans to reconstruct the agent's model [74], [80], [81]. This problem is the opposite of Inverse Reinforcement Learning (IRL) where we aim to make the RL imitate the user's model [81].

Huang *et al.* [81] applied this idea on autonomous driving cars. They built a model that helps users understand the agent objective function by showing a chosen set of agent decisions in selected scenarios. The goal of their method is to show the user scenarios that reflect the weight of different parameters in the agent optimization function. For example, in their autonomous car application, they choose scenarios where a safety based action (keeping the same lane) is different from an efficiency based action (bypass other cars), and hence based on the taken action the user can have an idea about the what parameters are more important in the agent objective function.

Moreover, Lage *et al.* [74], [80] used the reconstruction accuracy to evaluate different summary extractions methods generated by simulation, and then conducted experiments that involve humans to test their methods on the real world.

3) INTERESTINGNESS SUMMARY

Instead of summarizing the agent's policy with a list or a sequence of one data type as in the previous summarization

methods, multiple data items are collected from the interaction between a POMDP based agent and its environment. Then we use these generated items to deduce what is called *interestingness elements* [82]. Interestingness elements can be defined as some important measurements about the agent and its environment that are collected before (environment analysis), during (interaction analysis), and after (meta analysis) the interaction with the environment. The collected data are used to summarize both the agent behavior and its environment characteristics. We can describe the steps followed to generate these items as follows:

1) **Environment Analysis:** This step is done before starting the learning process where the certainty of transitions and observations (whether transition from a specific state/observation to another is certain or not) are recorded. Moreover, it includes reward analysis where any pair of state-action that results in a relatively high or low reward is recorded.
2) **Interaction Analysis:** This step contains the information that is collected during the interaction with the environment. Multiple measurements are collected from the observations of the agent such as: its observation coverage with respect to the observation space to measure the quality of exploration, actions coverage to reflect the level of interaction, the distribution of the recorded observations, and which observations are more frequent and which are rare. In addition, data about the correlation between observations and features, and between observations and actions are recorded. Statistical measures about the value function are also collected such as the outliers and the mean prediction error.
3) **Meta Analysis:** This step is performed on the trained agent. Data are collected about the associations between the value function and the observations, the pattern of actions sequence, and observations that lead to low/high value function. A contradiction analysis is conducted to show the situations where the agent took unexpected actions. This is done by highlighting the observations that have values different from their rewards, actions that are different from their values, and domain-specific sub-goals that are not reached.

One of the challenges of this method is that it generates a relatively huge amount of information. Since one of the main objectives is to help humans understand the behavior of the model, we think this method can be improved by conducting experiments that involve interaction with humans similar to the one carried out by the authors of HIGHLIGHTS [76]. Then we can see how the generated elements achieve their goal.

As an extension to this method, Sequeira and Gervasio [83] created a visual explanation in a short video clip that contains these interestingness elements. This contribution can help humans comprehend the huge number of items generated from the original method. The authors [83] also asked users to evaluate the visual form of the generated explanations, and found that if we have different agents, we need different set of interestingness elements to help users understand them more clearly.

D. CUSTOM LEARNING MODELS

In these methods, a new inherently interpretable learning model is proposed. Note that decision trees can also be classified under this category. However, since decision trees are generic and used to interpret other ML models, in addition to the fact that they have a lot of sub-categories, we put them in a separate section. For this category, we list only the inherently interpretable methods which were built specifically for RL.

1) ALGEBRAIC LANGUAGES

Maes *et al.* [84] proposed a new model that represents MDP-based RL policies in a human-readable language. The steps to build the model can be described as follows:

1) Selecting a custom human-readable language that consists of a limited set of operators and terminal symbols. The operators of the chosen language are used to explain the scores of state-actions.
2) Defining a metric to evaluate the language selected in the first step.
3) Applying the chosen metric to evaluate the interpretability of the policy we need to explain.

The authors defined a language that consists of either binary or unitary operators applied on a limited set of variables and constants. Then, the interpretability of the model is defined by the total number of variables, constants and operators that were used to represent the policy. For the operators, the authors used *min* and *max* functions, in addition to some simple algebraic operators such as plus, minus and square root. The variables are the states and actions in the RL policy we want to explain. In order to come up with the optimal policy in this representation, they used Monte Carlo simulations to generate the search space. Then after that, the problem is formulated as a multi-armed bandit problem. The authors did not include samples for the generated explanations or conduct any type of experiments or surveys that involve non-expert judgement of the interpretability of their approach. We think this part should be studied more deeply. Although the used language was described as human-readable, it is actually an algebraic based language, which can be considered human-readable only if its statements have a relatively short length in a way that enables humans to understand them easily.

Similarly, Hein *et al.* [85] introduced another algebraic based method for RL interpretation but with using genetic algorithms. The authors collected data of four dimensions that include: current state, action, next state and the received reward, and then trained their algorithm for two Atari games: Mountain Car and Cart-Pole. They also showed an example for their generated interpretable policies, which consists of linear equations accompanied with if/else rules. This explanation is very similar to the these generated by decision trees-based policies explained in Section VII-A. However, we

think more data is needed in order to see whether their method can be applied to more complex problems.

2) CUSTOM DOMAIN PROGRAMMING LANGUAGE

For this method, instead of using a neural network as a learning model, we first design a custom domain programming language [86], [87]. we have to design a language with instructions as a first step. Some examples of these instructions are: *change the car's direction* and *increase/decrease the speed*. Then, the generated language is used to fit a program that maximizes the policy reward. In other words, it is like making the computer uses a dataset to write an interpretable algorithm automatically. There are two challenges that needs to be solved:

1) How to design a custom and interpretable programming language for the application domain.
2) How to optimize the policy within the designed language.

The authors of this method [86], [87] solved the second challenge by using imitation learning. However, designing a custom domain language for each new domain is still needed. In addition to being human-readable, this approach also makes it easier for the user to add his own knowledge or customize the learned policy by altering the generated code after the learning phase.

3) FORMAL SPECIFICATION LANGUAGE

Li and Xiao [88] proposed a formal method for RL interpretation that allows us to prevent the RL agent from doing unwanted behaviors. They perform this by adding more complex constrains to the reward function using temporal logic. As a result, instead of using one reward function that might lead to a non-desired output (e.g., a cleaning robot might learn to mess the place first before cleaning it), these new constraints are stated explicitly in the reward function to prevent such behaviors. However, the altered reward function has a non-markovian nature (does not satisfy Markov assumption). To solve this problem, the authors introduced a new policy search algorithm based on temporal logic that utilizes constraint optimization to get the optimal policy and proved to satisfy the added constraints. The authors tested their method on a vehicle navigation application with adding obstacle avoidance constraints explicitly, instead of assigning a negative reward, and it outperformed another agent trained on a normal reward function. The main benefit of this method is that it makes the model more explainable by showing the motivation objectives explicitly instead of hiding them behind a negative reward. However, these constraints have to be defined manually by domain experts which might limit the applicability of this method in different fields.

4) INTERPRETABLE HIERARCHICAL RL

Hierarchical RL (HRL) tries to solve RL problems by converting the optimization goal into smaller sub-goals first, and then the optimization is done for each sub-goal separately [89]. In other words, instead of having only one policy that maps states to actions, we have multiple local policies where each one is responsible for achieving a specific sub-goal [90], [91], in addition to a global policy that learns these sub-goals. Sometimes, the generated sub-goals are divided again into smaller sub-goals which results in more than two-levels hierarchical reinforcement learning [91]. If the sub-goals are understandable to humans and can be used to explain the agent's behaviour, they are considered a type of RL interpretation. Note that not all the hierarchical RL methods are used for interpretation, and hence we survey in this section the methods that have interpretable sub-goals.

One of the problems of hierarchical RL is that the global policy must use only the list of available local policies to achieve the goal. Shu and Socher [91] introduced a hierarchical learning model that solves this challenge by adding the sub-goals dynamically by humans during the learning in a setup called weak supervision. In addition, the newly added sub-goal can also be decomposed into other new sub-goals. Although the generated explanations might be useful for understanding the agent's policy, we need them to be created by actual domain experts which challenges the generalization of this method across different applications, and increases the cost of implementation.

Beyre *et al.* [92] proposed a similar algorithm called **Dot-to-Dot** to apply interpretable HRL on robotic systems. Their implementation is based on two agents: a high level agent that learns the environment and task dynamics (the global policy) and a low level agent that performs the raw law-level actions (the local policies). In their application, a robotic arm that needs to reach a specific point by moving in different directions, is explained by training it to reach intermediate points (sub-goals) which lead to the final goal. The sub-goals are trained automatically and do not need to be defined in advance.

We can also think of state space abstraction methods as some sort of interpretable HRL. However, the difference here instead of dividing the learning objective into multiple sub-goals, we convert the state space into a smaller space, and then we make the policy map the actions to this new domain. This approach was followed by Akrour *et al.* [93]. They performed this by clustering the states into new interpretable centers.

5) RECONSTRUCTION

This method is used to interpret DQN architecture and is applied on different Atari games that have image inputs [94]. In addition to the normal architecture used to generate the Q values in DQN, more layers (deconvoutional layers) are added that are able to reconstruct the inputs again and the reconstruction error (the quality of the reconstructed images) is used as one of the parameters to learn the whole network. Before applying the DQN outputs to the reconstruction network, a latent space is first created which represents the input source of the reconstruction network. The reconstructed image together with the latent space is used to understand

which parts of the image were effective in taking decisions by the agent.

E. NATURAL LANGUAGE EXPLANATIONS

One of RL interpretation methods is to explain the policy or some parts of it in human's natural languages. Note that the methods classified under this category are different from the custom models which are based on a human-readable language that we explained in Section VII-D. In this approach, instead of using an inherently human readable language-based model, we add the natural language explanations to the already existent black box models. Explanations can be made either to explain an individual decision to justify why a specific action is chosen (local template-based explanations), the entire policy, or to answer user's specific queries. Natural language explanations can also be made to justify the selection of a specific decision instead of another one (contrastive explanations).

1) TEMPLATE-BASED EXPLANATIONS

This approach is used to explain individual decisions of MDP-based RL policies. The goal is to explain why a specific action was recommended by an RL agent by filling a natural language-based template. Wang *et al.* [95] built a framework that generates template-based explanations to interpret POMDP based robotic systems. Their system communicates the different components of the POMDP systems (e.g., state space, and action space) in human readable formats. For example, it can explain the observed state to give the user an idea about its sensors view.

Similarly, Khan *et al.* [96], [97] first created a domain independent template. If we need to explain a specific action, a template is filled by the states and scenarios that make taking this action more likely. The next step is to fill this template with the specific states and scenarios by running the algorithm from the starting point (the action we want to explain) until the final goal. Moreover, the authors of this method [96] also proposed a framework for multiple template generation that claimed to be minimal and sufficient. According to their definition, an explanation (a group of templates) is sufficient if it can justify the optimality of a recommendation without need to use more templates. On the other hand, an explanation is said to be minimal if it includes the minimum possible number of templates. This method was applied to two different problems: 1) advising students in course selection, and 2) helping people who have dementia to wash their hands. Since this explanation method is similar to the task advisor in course selection task, the authors asked student advisors to evaluate the generated explanations and their responses were mainly positive. However, they did not conduct a quantitative evaluation.

Dodson *et al.* [98], [99] applied a similar model on academic advising. However, their model is a mixture of domain independent and domain specific methodologies. The explanation model is domain independent, while domain knowledge is represented in the generated language (system's interface) and the explanation inputs (data sources). This interpretation method explains why a specific action is recommended by the agent's policy in terms of both past and predicted data. The authors designed their system in a modular way to contain the following parts:

1) **Model Based Explanation Module:** explains the recommended action in terms of the best possible actions it leads to.
2) **Case-Based Explanation Module:** explains the recommended action in terms of the past performance.
3) **Natural Language Generator Module:** this module is responsible for generating domain-dependent natural language expressions using the outputs of the previous modules.

2) QUERY-BASED EXPLANATIONS

A query is a question asked by the user that explains the agent's policy or its individual decisions. Explanations are provided through answering these queries. Hayes and Shah [100] designed a method that answers the following types of questions for both MDP based and $Q-$ learning RL systems.

1) What are the environmental conditions of performing a specific action?
2) State the actions that the agent performs for a given set of environmental conditions.
3) Why the agent did **not** take a given action.

The authors were able to answer these questions by performing the following steps:

1) Write the query in a natural language. For example, we can make the following query to a self-driving car agent: "When do you turn left?".
2) Map the query to a template (for example, "When do you do{action}?")
3) Investigate the state space in order to find the states where this action is likely.
4) Convert the list of these states (state map) to a natural language representation that is shown to the user as the final explanation.

The authors tested their method on three different robot control applications and showed their explanations. They also compared some samples from their explanations with expert explanations which showed some similarities. However, it is hard to evaluate this part quantitatively since it includes a natural language input. A different type of query based explanations is contrastive explanations. This method proposed by van der Waa *et al.* [101] gives an explanation to justify why an RL agent took a specific action instead of an alternative. In other words, an answer to a contrastive question such as *why the agent performs action x instead of y* is answered as an explanation to the agent's decision. Before answering this question, first the state space is transformed into another domain that is more understandable to humans. For example, in a maze game we transform the x, y coordinates into a more meaningful location descriptions, such as *near the goal* and

near the monster. Also, a similar transformation is applied on the reward function (e.g., instead of using the continuous scores, we can change them into two discrete values: positive and negative only). After this step, the contrastive question is used to generate a foil policy that is very similar to the agent's original policy, but it does the contrastive behavior stated in the question. Then this new constructed policy is used to generate the most likely actions and states that will be investigated by the agent. Finally, these states and actions are combined together to construct the final textual explanation.

F. MODEL RECONCILIATION

These methods are based on the following assumptions: 1) humans have their own conceptual model M_h of the agent's model M_a, which might be different from the actual agent's model, and 2) their ability to predict the agent's behavior is limited [102]. This approach was also used for classical planning problems that uses algorithms such as A*. For example, Korpan and Epstein [103] proposed utilizing M_a and M_h to generate natural language explanations to bridge the gap between the two models. The objective of these methods is to help humans understand the agent decisions starting from their mental model M_h. We can divide this approach into two main sub-categories: certain and uncertain.

1) CERTAIN MODEL RECONCILIATION

This method is built on the assumption that we have a complete knowledge about the human's model M_h [27] -just like the agent model M_a - that is described in the Planning Domain Definition Language (PDDL). The authors of this method [27] called the process of making humans understand M_a through this type of explanation *model reconciliation*. The human expects some action from the agent based on their M_h. When this expectation is different from the actual action, the agent tries to correct M_h to match M_a and decrease the difference between the two models M_h and M_a. This correction is given also in PDDL format. For example, it may tell the user that the previous action has some preconditions, and that is why action x is performed before what they expected [104].

2) UNCERTAIN MODEL RECONCILIATION

This method proposed by Sreedharan *et al.* [27] is an extension to the previous method where the full knowledge of the human model is no longer required i.e., it generates an explanation despite having different possible combinations of M_h. The provided explanations are called **conformant explanations** since they work for all combinations.

VIII. CONCLUSION

In this paper, we gave a high level overview of the approaches followed in RL interpretations, discussed categorization metrics used to classify interpretation methods, and applied them on the methods we reviewed. Some of these methods were borrowed from the interpretability literature of machine learning such as saliency maps, while others were proposed mainly to interpret RL policies such as the custom learning models.

Categorization of interpretation methods is difficult. The ambiguity of terms such as ''interpretation'', ''explanation'', ''justification'', and others muddy the waters. We tried to use ''interpretation'' to describe the objective of the methods, and ''explanations'' to describe the individual outputs of the different approaches. In addition, we described the applications tested by the method's authors, and the majority of these methods were applied on Atari games. It stands to reason this property can be utilized to build a benchmark platform for RL interpretability which enables comparisons between methods on the same problem. It is understandable this might not be possible when the domains are very different. However, even benchmarking for homogeneous groups of interpretation methods is absent. For example we do not have a benchmark platform for saliency maps that target explaining the same kind of RL policies. The subjective part of interpretability is mainly measured by surveys. Different surveys have been adopted by different authors, and we think an added unified layer of output (domain experts and non-experts) to the evaluation process will be beneficial for the advancement of this field.

REFERENCES

[1] H. Chia, ''In machines we trust: Are robo-advisers more trustworthy than human financial advisers?'' *Law, Tech. Hum.*, vol. 1, p. 129, 2019.
[2] D. Nikulin, A. Ianina, V. Aliev, and S. Nikolenko, ''Free-lunch saliency via attention in atari agents,'' 2019, *arXiv:1908.02511*. [Online]. Available: http://arxiv.org/abs/1908.02511
[3] A. Weller, ''Transparency: Motivations and challenges,'' in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019, pp. 23–40.
[4] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, ''Explaining explanations: An overview of interpretability of machine learning,'' in *Proc. IEEE 5th Int. Conf. Data Sci. Adv. Analytics (DSAA)*, Oct. 2018, pp. 80–89.
[5] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, ''Explaining explanations: An overview of interpretability of machine learning,'' 2018, *arXiv:1806.00069*. [Online]. Available: http://arxiv.org/abs/1806.00069
[6] S. Mohseni, N. Zarei, and E. D. Ragan, ''A multidisciplinary survey and framework for design and evaluation of explainable AI systems,'' 2018, *arXiv:1811.11839*. [Online]. Available: http://arxiv.org/abs/1811.11839
[7] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, ''A survey of methods for explaining black box models,'' *ACM Comput. Surveys*, vol. 51, no. 5, pp. 1–42, Jan. 2019.
[8] C. Molnar, ''Interpretable machine learning,'' *Lulu.com*, 2019.
[9] A. Bibal and B. Frénay, ''Interpretability of machine learning models and representations: An introduction,'' in *Proc. ESANN*, 2016, pp. 1–6.
[10] W. James Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, ''Interpretable machine learning: Definitions, methods, and applications,'' 2019, *arXiv:1901.04592*. [Online]. Available: http://arxiv.org/abs/1901.04592
[11] O. Biran and C. Cotton, ''Explanation and justification in machine learning: A survey,'' in *Proc. IJCAI Workshop Explainable AI (XAI)*, vol. 8, 2017, p. 1.
[12] A. Adadi and M. Berrada, ''Peeking inside the black-box: A survey on explainable artificial intelligence (XAI),'' *IEEE Access*, vol. 6, pp. 52138–52160, 2018.
[13] M. A. Ahmad, A. Teredesai, and C. Eckert, ''Interpretable machine learning in healthcare,'' in *Proc. IEEE Int. Conf. Healthcare Informat. (ICHI)*, Jun. 2018, pp. 559–560.
[14] Q.-S. Zhang and S.-C. Zhu, ''Visual interpretability for deep learning: A survey,'' *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 1, pp. 27–39, Jan. 2018.

[15] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8827–8836.

[16] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 6541–6549.

[17] T. M. Mitchell *et al.*, *Machine Learning*, vol. 45, no. 37. Burr Ridge, IL, USA: McGraw-Hill, 1997, pp. 870–877.

[18] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, Sep. 2013.

[19] A. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electron. Imag.*, vol. 2017, no. 19, pp. 70–76, Jan. 2017.

[20] X. Hu, P.-Y. Hsueh, C.-H. Chen, K. M. Diaz, F. Parsons, I. Ensari, M. Qian, and Y.-K. Cheung, "An interpretable health behavioral intervention policy for mobile device users," *IBM J. Res. Develop.*, vol. 62, no. 1, pp. 1–4, Jan./Feb. 2018.

[21] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

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

[24] E. Akhtar and S. Farrukh, *Practical Reinforcement Learning: Develop Self-Evolving, Intelligent Agents With OpenAI Gym, Python and Java*. Birmingham, U.K.: Packt Publishing, 2017.

[25] L. D. Pyeatt and A. E. Howe, "Decision tree function approximation in reinforcement learning," in *Proc. 3rd Int. Symp. Adapt. Syst., Evol. Comput. Probabilistic Graph. Models*, vol. 2, 2001, pp. 70–77.

[26] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine learning interpretability: A survey on methods and metrics," *Electronics*, vol. 8, no. 8, p. 832, Jul. 2019.

[27] T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati, "Plan explanations as model reconciliation: Moving beyond explanation as soliloquy," in *Proc. IJCAI Int. Joint Conf. Artif. Intell.*, vol. 1802, 2017, pp. 156–163.

[28] D. Kazhdan, Z. Shams, and P. Liò, "MARLeME: A multi-agent reinforcement learning model extraction library," 2020, *arXiv:2004.07928*. [Online]. Available: https://arxiv.org/abs/2004.07928

[29] A. Papenmeier, G. Englebienne, and C. Seifert, "How model accuracy and explanation fidelity influence user trust," 2019, *arXiv:1907.12652*. [Online]. Available: http://arxiv.org/abs/1907.12652

[30] A. M. Roth, N. Topin, P. Jamshidi, and M. Veloso, "Conservative Q-improvement: Reinforcement learning for an interpretable decision-tree policy," 2019, *arXiv:1907.01180*. [Online]. Available: http://arxiv.org/abs/1907.01180

[31] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2494–2504.

[32] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: https://arxiv.org/abs/1503.02531

[33] A. Silva, T. Killian, I. D. J. Rodriguez, S.-H. Son, and M. Gombolay, "Optimization methods for interpretable differentiable decision trees in reinforcement learning," 2019, *arXiv:1903.09338*. [Online]. Available: https://arxiv.org/abs/1903.09338

[34] I. D. J. Rodriguez, W. T. Killian, S. Son, and C. M. Gombolay, "Interpretable reinforcement learning via differentiable decision trees," 2019, *arXiv:1903.09338*. [Online]. Available: http://arxiv.org/abs/1903.09338

[35] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Distal explanations for explainable reinforcement learning agents," 2020, *arXiv:2001.10284*. [Online]. Available: http://arxiv.org/abs/2001.10284

[36] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Explainable reinforcement learning through a causal lens," 2019, *arXiv:1905.10958*. [Online]. Available: https://arxiv.org/abs/1905.10958

[37] Y. Coppens, K. Efthymiadis, T. Lenaerts, and A. Nowe, "Distilling deep reinforcement learning policies in soft decision trees," in *Proc. IJCAI Workshop Explainable Artif. Intell. (IJCAI) Workshop Explainable Artif. Intell.*, Macao, China, T. Miller, R. Weber, and D. Magazzeni, Eds. Aug. 2019, pp. 1–6.

[38] S. Karakovskiy and J. Togelius, "The mario AI benchmark and competitions," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 55–67, Mar. 2012.

[39] G. Liu, O. Schulte, W. Zhu, and Q. Li, "Toward interpretable deep reinforcement learning with linear model u-trees," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Dublin, Ireland: Springer, 2018, pp. 414–429.

[40] M. Vasic, A. Petrovic, K. Wang, M. Nikolic, R. Singh, and S. Khurshid, "MoET: Interpretable and verifiable reinforcement learning via mixture of expert trees," 2019, *arXiv:1906.06717*. [Online]. Available: http://arxiv.org/abs/1906.06717

[41] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1177–1193, Aug. 2012.

[42] A. Brown and M. Petrik, "Interpretable reinforcement learning with ensemble methods," 2018, *arXiv:1809.06995*. [Online]. Available: http://arxiv.org/abs/1809.06995

[43] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," 2017, *arXiv:1704.02685*. [Online]. Available: https://arxiv.org/abs/1704.02685

[44] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013, *arXiv:1312.6034*. [Online]. Available: http://arxiv.org/abs/1312.6034

[45] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," 2014, *arXiv:1412.6806*. [Online]. Available: https://arxiv.org/abs/1412.6806

[46] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, no. 7, Jul. 2015, Art. no. e0130140.

[47] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," 2017, *arXiv:1703.01365*. [Online]. Available: https://arxiv.org/abs/1703.01365

[48] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "Smoothgrad: Removing noise by adding noise," 2017, *arXiv:1706.03825*. [Online]. Available: https://arxiv.org/abs/1706.03825

[49] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "A benchmark for interpretability methods in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 9734–9745.

[50] J. Seo, J. Choe, J. Koo, S. Jeon, B. Kim, and T. Jeon, "Noise-adding methods of saliency map as series of higher order partial derivative," 2018, *arXiv:1806.03000*. [Online]. Available: http://arxiv.org/abs/1806.03000

[51] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*. [Online]. Available: http://arxiv.org/abs/1511.06581

[52] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," 2017, *arXiv:1711.00138*. [Online]. Available: http://arxiv.org/abs/1711.00138

[53] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 3429–3437.

[54] Z. Yang, S. Bai, L. Zhang, and P. H. S. Torr, "Learn to interpret atari agents," 2018, *arXiv:1812.11276*. [Online]. Available: http://arxiv.org/abs/1812.11276

[55] P. Gupta, N. Puri, S. Verma, S. Singh, D. Kayastha, S. Deshmukh, and B. Krishnamurthy, "Explain your move: Understanding agent actions using focused feature saliency," 2019, *arXiv:1912.12191*. [Online]. Available: https://arxiv.org/abs/1912.12191

[56] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*. [Online]. Available: http://arxiv.org/abs/1409.0473

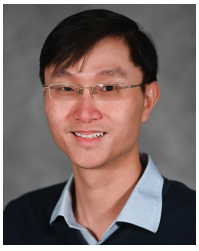
[57] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva, "Deep attention recurrent Q-Network," 2015, *arXiv:1512.01693*. [Online]. Available: http://arxiv.org/abs/1512.01693

[58] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp.*, 2015, pp. 1–52.

[59] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[60] S. Mousavi, M. Schukat, E. Howley, A. Borji, and N. Mozayani, "Learning to predict where to look in interactive environments using deep recurrent q-learning," 2016, *arXiv:1612.05753*. [Online]. Available: https://arxiv.org/abs/1612.05753
[61] R. Zhang, Z. Liu, L. Zhang, A. Jake Whritner, S. Karl Muller, M. Mary Hayhoe, and H. Dana Ballard, "AGIL: Learning attention from human for visuomotor tasks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 663–679.
[62] R. Zhang, C. Walshe, Z. Liu, L. Guan, K. S. Muller, J. A. Whritner, L. Zhang, M. M. Hayhoe, and D. H. Ballard, "Atari-HEAD: Atari human eye-tracking and demonstration dataset," 2019, *arXiv:1903.06754*. [Online]. Available: https://arxiv.org/abs/1903.06754
[63] A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro, "Show, attend and interact: Perceivable human-robot social interaction through neural attention Q-network," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 1639–1645.
[64] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9505–9515.
[65] A. Atrey, K. Clary, and D. Jensen, "Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning," 2019, *arXiv:1912.05743*. [Online]. Available: http://arxiv.org/abs/1912.05743
[66] J. Luo, S. Green, P. Feghali, G. Legrady, and Ç. Kaya Koç, "Visual diagnostics for deep reinforcement learning policy development," 2018, *arXiv:1809.06781*. [Online]. Available: http://arxiv.org/abs/1809.06781
[67] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1–13.
[68] J. Peng and R. J. Williams, "Incremental multi-step q-learning," in *Machine Learning Proceedings*. Amsterdam, The Netherlands: Elsevier, 1994, pp. 226–232.
[69] A. Dhurandhar, P.-Y. Chen, R. Luss, C.-C. Tu, P. Ting, K. Shanmugam, and P. Das, "Explanations based on the missing: Towards contrastive explanations with pertinent negatives," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 592–603.
[70] M. L. Olson, L. Neal, F. Li, and W.-K. Wong, "Counterfactual states for atari agents via generative deep learning," 2019, *arXiv:1909.12969*. [Online]. Available: http://arxiv.org/abs/1909.12969
[71] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.
[72] N. B. Zrihem, T. Zahavy, and S. Mannor, "Visualizing dynamics: From t-SNE to SEMI-MDPs," 2016, *arXiv:1606.07112*. [Online]. Available: https://arxiv.org/abs/1606.07112
[73] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding DQNs," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1899–1908.
[74] I. Lage, D. Lifschitz, F. Doshi-Velez, and O. Amir, "Exploring computational user models for agent policy summarization," 2019, *arXiv:1905.13271*. [Online]. Available: http://arxiv.org/abs/1905.13271
[75] O. Amir, F. Doshi-Velez, and D. Sarne, "Agent strategy summarization," in *Proc. 17th Int. Conf. Auto. Agents MultiAgent Syst.*, 2018, pp. 1203–1207.
[76] D. Amir and O. Amir, "Highlights: Summarizing agent behavior to people," in *Proc. 17th Int. Conf. Auton. Agents MultiAgent Syst.*, 2018, pp. 1168–1176.
[77] S. H. Huang, K. Bhatia, P. Abbeel, and A. D. Dragan, "Establishing appropriate trust via critical states," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 3929–3936.
[78] N. Topin and M. Veloso, "Generation of policy-level explanations for reinforcement learning," 2019, *arXiv:1905.12044*. [Online]. Available: http://arxiv.org/abs/1905.12044
[79] A. Zien, N. Kraemer, S. Sonnenburg, and G. Raetsch, "The feature importance ranking measure," 2009, *arXiv:0906.4258*. [Online]. Available: https://arxiv.org/abs/0906.4258
[80] I. Lage, D. Lifschitz, F. Doshi-Velez, and O. Amir, "Toward robust policy summarization," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 2081–2083.
[81] S. Huang, D. Held, P. Abbeel, and A. Dragan, "Enabling robots to communicate their objectives," in *Robotics: Science and Systems XIII*. Jul. 2017.
[82] P. Sequeira and M. Gervasio, "Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations," *Artif. Intell.*, vol. 288, Nov. 2020, Art. no. 103367, doi: 10.1016/j.artint.2020.103367.
[83] F. Sado, C. Kiong Loo, M. Kerzel, and S. Wermter, "Explainable goal-driven agents and robots - a comprehensive review and new framework," 2020, *arXiv:2004.09705*. [Online]. Available: http://arxiv.org/abs/2004.09705
[84] F. Maes, R. Fonteneau, L. Wehenkel, and D. Ernst, "Policy search in a space of simple closed-form formulas: Towards interpretability of reinforcement learning," in *Proc. Int. Conf. Discovery Sci.* Lyon, France: Springer, Oct. 2012, pp. 37–51.
[85] D. Hein, S. Udluft, and T. A. Runkler, "Interpretable policies for reinforcement learning by genetic programming," 2017, *arXiv:1712.04170*. [Online]. Available: https://arxiv.org/abs/1712.04170
[86] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," 2018, *arXiv:1804.02477*. [Online]. Available: http://arxiv.org/abs/1804.02477
[87] A. Verma, "Verifiable and interpretable reinforcement learning through program synthesis," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 9902–9903.
[88] X. Li, "A formal methods approach to interpretability, safety and composability for reinforcement learning," Ph.D. dissertation, Dept. Mech. Eng., Boston Univ., Boston, MA, USA, 2020.
[89] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, nos. 1–2, pp. 41–77, 2003.
[90] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 3303–3313.
[91] T. Shu, C. Xiong, and R. Socher, "Hierarchical and interpretable skill acquisition in multi-task reinforcement learning," 2017, *arXiv:1712.07294*. [Online]. Available: http://arxiv.org/abs/1712.07294
[92] B. Beyret, A. Shafti, and A. A. Faisal, "Dot-to-Dot: Explainable hierarchical reinforcement learning for robotic manipulation," 2019, *arXiv:1904.06703*. [Online]. Available: https://arxiv.org/abs/1904.06703
[93] R. Akrour, D. Tateo, and J. Peters. *Towards Reinforcement Learning of Human Readable Policies*. [Online]. Available: https://www.ias.informatik.tu-darmstadt.de/uploads/Team/RiadAkrour/decodeml2019.pdf
[94] R. M. Annasamy and K. Sycara, "Towards better interpretability in deep q-networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4561–4569.
[95] N. Wang, D. V. Pynadath, and S. G. Hill, "Trust calibration within a human-robot team: Comparing automatically generated explanations," in *Proc. 11th ACM/IEEE Int. Conf. Human-Robot Interact. (HRI)*, Mar. 2016, pp. 109–116.
[96] O. Z. Khan, P. Poupart, and J. P. Black, "Minimal sufficient explanations for factored Markov decision processes," in *Proc. 19th Int. Conf. Automated Planning Scheduling*, 2009, pp. 1–7.
[97] O. Z. Khan, *Policy Explanation and Model Refinement in Decision-Theoretic Planning*. Waterloo, ON, Canada: Univ. of Waterloo, 2013.
[98] T. Dodson, N. Mattei, and J. Goldsmith, "A natural language argumentation interface for explanation generation in Markov decision processes," in *Proc. Int. Conf. Algorithmic Decis. Theory*. Piscataway, NJ, USA: Springer, 2011, pp. 42–55.
[99] T. Dodson, N. Mattei, J. T. Guerin, and J. Goldsmith, "An English-language argumentation interface for explanation generation with Markov decision processes in the domain of academic advising," *ACM Trans.s Interact. Intell. Syst.*, vol. 3, no. 3, p. 18, 2013.
[100] B. Hayes and J. A. Shah, "Improving robot controller transparency through autonomous policy explanation," in *Proc. ACM/IEEE Int. Conf. Hum.-Robot Interact.*, Mar. 2017, pp. 303–312.
[101] J. van der Waa, J. van Diggelen, K. van den Bosch, and M. Neerincx, "Contrastive explanations for reinforcement learning in terms of expected consequences," 2018, *arXiv:1807.08706*. [Online]. Available: http://arxiv.org/abs/1807.08706
[102] S. Penney, J. Dodge, C. Hilderbrand, A. Anderson, L. Simpson, and M. Burnett, "Toward foraging for understanding of StarCraft agents: An empirical study," in *Proc. 23rd Int. Conf. Intell. User Interfaces*, 2018, pp. 225–237.
[103] R. Korpan, S. L. Epstein, A. Aroor, and G. Dekel, "WHY: Natural explanations from a robot navigator," 2017, *arXiv:1709.09741*. [Online]. Available: http://arxiv.org/abs/1709.09741
[104] S. Sreedharan and S. Kambhampati, "Handling model uncertainty and multiplicity in explanations via model reconciliation," in *Proc. 28th Int. Conf. Automated Planning Scheduling*, 2018, pp. 1–10.

ALNOUR ALHARIN (Member, IEEE) is currently pursuing the data science master's degree (area of specialization is statistics). He is also a Machine Learning Engineer and a Researcher with a solid experience in software engineering, and a strong background in computer science. His current research interests include machine learning (especially reinforcement learning) interpretability and car traffic estimation through computer vision (objects detection and tracking) in smart city applications. He has extensive experience in multiple programming languages especially Python, C/C++, and Java. He has also a full working knowledge both in back-end and Android development through the implementation of several projects.



THANH-NAM DOAN received the bachelor of engineering degree in computer science from the Ho Chi Minh City University of Technology, in 2011, and the Ph.D. degree from Singapore Management University, in 2018, where he studied about the two new latent features of location named: neighborhood competition and area attractiveness. After his doctoral training, he spent one year as a Postdoctoral Researcher in the State University of New York, Albany, honing his expertise in the characterization of deep learning for recommender systems and educational data mining. He has been joining The University of Tennessee at Chattanooga since August 2019 under the supervision of Dr. Sartipi. In UTC, he focuses on working with transportation data and his goal is to apply big data techniques, data mining, and machine learning toward more cost-effective public transportation as well as more efficient way to manage traffic lights. He is also working with other students to build an interpretable machine learning models which could help people to understand the underlying process without much mathematical knowledge.

MINA SARTIPI (Senior Member, IEEE) received the B.S. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2001, and the M.S. and Ph.D. degrees in electrical and computer engineering from Georgia Tech, in 2003 and 2006, respectively. She is currently the Director of the Center for Urban Informatics and Progress (CUIP). She is also a UC Foundation Professor with the Computer Science and Engineering Department, where she leads the Smart Communications and Analysis Lab (SCAL). Her research interests include communications and data science, in particular advanced wireless communications and data analysis for smart healthcare and urban futures. Dr. Sartipi has been a member of the board of directors for Variable, Inc., Chattanooga, TN, USA, since 2013, and The Enterprise Center, Chattanooga, TN, USA, since 2017. In 2008, she was named UC Foundation Assistant Professor. This award was given to her based on her research activities and students evaluating her teaching. She was awarded the UTC Outstanding Faculty Research and Creative Achievement Award in 2016. She had been awarded the Best Researcher with the Department of CSE and the College of CECS in 2010, 2013, 2014, and 2015. She has served as the Technical Program Chair of conferences in the areas of wireless communications and networking. Additionally, she has been called on to review several articles on data compression and error control coding for various IEEE, ACM, and EURASIP conferences and journals in the past several years.