Practical Control Design for the Deep Learning Age:
Distillation of Deep RL-Based Controllers

Nathan Dahlin, Kevin Chang, Krishna Chaitanya Kalagarla, Rahul Jain and Pierluigi Nuzzo

Abstract—Deep Reinforcement Learning (RL) methods produce often performance-leading controllers. Yet, in many applications (e.g., when safety and assurance are critical), they cannot be deployed. Deep Neural Nets (DNN) controllers are nearly impossible to verify and their behavior hard to interpret. In this paper, we first show, through a simple example, that there exist soft decision tree (SDT) controllers that are equivalent to neural network (NN) controllers in terms of input-output behavior. Based on imitation learning, we then show via three OpenAI gym environment case studies that it is possible to distill high-performance DNN controllers into non-DNN controllers (e.g., leveraging decision tree and support vector machine architectures) that, while sacrificing performance just a bit, can be simpler, more interpretable, hence amenable to verification and validation. We thus propose the natural control design paradigm that leverages the power of deep RL methods to design reference DNN controllers, and then distill them into non-DNN controllers which can be validated and deployed in real systems. Finally, we identify some distillation metrics that can be useful in assessing the quality of the distilled controllers.

I. INTRODUCTION

Deep Reinforcement Learning (RL) methods [27] have proved remarkably effective at solving some of the toughest control problems. For many benchmark problems, deep RL methods-based controllers lead the leaderboards. And yet, when it comes to deployment of such controllers in real systems, particularly safety-critical applications such as flight control and autonomous driving, there is hesitancy. This happens because, despite their superior empirical performance, the properties of such controllers are challenging to verify [31], [29]. They are like black boxes and their behavior does not seem amenable to interpretability [5], [18]. Furthermore, in some applications, the size and complexity of deep neural nets (DNNs) can preclude their use in real time due to limited computational resources [9], [26].

This situation raises the following key questions: 1) Do there even exist non-DNN controllers that are equivalent (in terms of input-output properties) to DNN controllers designed using deep RL? Can we construct such controllers? 2) Can we design non-DNN controllers in a scalable manner that approximate the performance of DNN controllers while being verifiable and more interpretable?

This paper is partly inspired by the work of Frosst and Hinton [16] that asked similar questions in the context of supervised learning, and that of Bastani et al. [6], aiming at interpretability of DNN controllers via distillation. It is well known that alternative decision-making paradigms including decision trees and kernel methods share the universal approximation capacity of neural networks (NNs) [43]. Additionally, multiple studies [3], [6] attribute the efficacy of DNNs not to an inherently richer representative capacity over other architectures, but rather to the many regularization techniques which currently facilitate DNN training. Therefore, while it may not always be clear how to obtain alternative controllers with performance similar to DNNs, it is theoretically possible, and motivates the search for them.

In this paper, we make the following contributions. Through a simple example, we first show that, for a simple single-hidden-layer NN, there exists an equivalent soft decision tree (SDT) with the same input-output properties. The construction can be generalized but may not result in a scalable SDT (for verification or implementation purposes). This motivates the search for architectures that are more scalable. We show that using a standard machine learning library, sklearn, we can distill DNN controllers designed using the deep Q-network (DQN) algorithm [27], one of the best performing deep RL algorithms, into decision tree and Gaussian-kernel support vector machine (SVM)-based controllers. We consider three benchmarks from the OpenAI gym environments to illustrate the results: MountainCar-v0, CartPole-v0, and CarRacing-v0. We identify key metrics by which distillation performance can be assessed, and show that the distilled non-DNN controllers are approximations to the DNN controllers in terms of the distillation metrics.

The implications of our work are the following: 1) For a given NN controller, there exists a non-NN controller (as an SDT) with exactly the same input-output properties. Thus, certain properties (e.g., safety) of the NN controller can be verified on the non-NN controller, which may be less expensive. 2) When exactly equivalent non-DNN controllers are not achievable in a scalable manner, we can obtain suitable approximations to the DNN controllers in terms of decision trees and SVMs. Such non-DNN distilled controllers sacrifice the performance a bit of the reference DNN controllers, but can be more amenable to verification and validation, implementation, and interpretability. Therefore, we can first use deep RL methods to design reference controllers, and then distill them to non-DNN architectures, verify the control properties of the distilled controllers, and implement them in real systems.
II. RELATED WORK

Distillation is the transfer of behavior or learned knowledge for a given problem or task from one model to another. This is related to, but distinct from, model compression, which aims at reduced complexity models of the same structure [36].

A teacher-student paradigm is central to the distillation procedure [8]. For example, Ba and Caruana [3] propose to train shallow NNs to achieve performance comparable to state-of-the-art DNNs, where the shallow NNs are trained to imitate the DNNs, instead of learning directly from the original training data. Training a student NN by using a weighted combination of the correct labels and the output soft labels (class probabilities) generated by a teacher network was showed to help transfer knowledge to the student with improved student performance [19]. This approach was also extended to more complex, multi-class object tasks [9]. By introducing SDTs, which essentially feature a single layer perceptron at each inner node, Frosst and Hinton demonstrated that training these SDTs on data generated by an expert DNN improves performance over training directly on the labeled data [16].

Distillation has also been attempted in RL problems. With the policy distillation technique, an RL agent’s policy can be extracted to a smaller, more efficient network that still performs at the expert level [40]. The DAgGER algorithm [39] has been augmented by making use of the expert’s Q-function to extract a series of policies, the best of which is selected based upon using expert’s Q-function. Finally, closest to our experimental work, Coppens et al. [10] studied how SDTs [16] can be used to explain the behavior of expert DNNs in an RL setting. We include standard decision trees as implemented in the sklearn package [35] as a baseline target architecture.

To date, the literature is scarce on approximation of DNNs via SVMs or kernel machines (KMs). Closest to our work, Fukui et al. [17] examine a similar distillation paradigm in the supervised learning setting, with CIFAR-10 image classification as the target application. Qian et al. [38] abstract the intermediate layers of NNs as kernel functions, and attempt to distil smaller NNs from a reference by using the distance between approximations of teacher and student kernel representations as a loss function, with CIFAR-100 as a target application. In contrast, we use distillation from NNs to KMs specifically targeting RL environments.

III. PRELIMINARIES

We begin with some basic background for the sake of completeness.

a) Imitation Learning: Our basic methodology for distillation rests on use of imitation learning (IL) methods [39] to train our student KMs to imitate the complex DNN experts. IL trains a classifier or regressor to predict the behavior of an expert based on a dataset of observation (state) and action trajectories produced by the expert. While extensions of this basic approach which account for discrepancies between the distributions of states visited by the expert and student are available [39], we employ the basic IL approach here and leave exploration of more advanced techniques to future research.

b) KMs and Multiclass Classification: KMs (also known as SVMs) make predictions on unlabeled inputs based upon weighted comparisons with a collection of labeled training data points. Thus, KMs provide interpretability in the sense that the decision made for a given input sample can be explained in terms of the contribution of each support vector, i.e., how similar the input is to each support vector. The similarity between new inputs and support vectors is calculated based on a function known as the kernel. In the case of binary classification, the classifier is determined by solving an optimization problem of the following form

\[
\min_{w,b,\zeta} \frac{1}{2}w^Tw + C \sum_{i=1}^{n} \zeta_i \\
\text{s.t. } y_i(w^T \varphi(x_i) + b) \geq 1 - \zeta_i, \\
\zeta_i \geq 0, \quad i = 1, \ldots, n,
\]

where \(x_i\) is a training sample with corresponding label \(y_i \in \{-1, +1\}\), \(\varphi()\) is a feature map and \(n\) gives the training sample size.

The solution to the above problem yields a label prediction for a new input \(x\) of the form [2]

\[
\hat{y}(x) = \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i k(x_i, x) + b \right),
\]

where \(\alpha_i \in \mathbb{R}_+ \forall i\), and \(b\) are the dual coefficients (also referred to as weights) to constraints (1), and offset, respectively, and \(k\) is a real-valued kernel function. Input samples corresponding to weights satisfying \(\alpha_i > 0\) are known as support vectors. Any function which corresponds to an inner product in some inner product space constitutes a valid kernel. In this work we focus on the commonly used Gaussian or radial basis kernel, given by \(k(x_i, x) = \exp(-\gamma\|x_i - x\|^2) = \langle \varphi(x_i), \varphi(x) \rangle\), where \(\gamma > 0\) is an input parameter. The weights and offset are determined by solving the above multiobjective optimization problem, which maximizes the margin between input classes while penalizing misclassifications with parameter \(C\).

In two of the case studies we present, the number of actions available to controllers is greater than two, so it is necessary for the surrogate KMs to perform multiclass classification on input environment states. While there are several proposals to extend the binary classification framework presented above to multiclass classification [11], [44], we use the straightforward one-vs-one approach as implemented in sklearn. This approach decomposes the multiclass classification problem into multiple binary classification problems and aggregates the results via a voting scheme. Binary classifier confidence levels are used to break ties between labels [35].
We consider the single-hidden-layer NN in Figure 1, with inputs $x = (x_0, x_1)^T \in \mathbb{R}^2$, where $v^T$ denotes the transpose of $v$. Let $L$ be the number of layers and $N^l$ the number of neurons in the $l$-th layer. In our case, we have $L = 3$, including a two-neuron input layer ($N^1 = 2$), a single neuron inner layer ($N^2 = 1$), and a two-neuron output layer ($N^3 = 2$). The nodes in the (non-input) layer $l$, $l > 1$, are fully connected to the previous layer. Edges and nodes are labelled by weights and biases, denoted by $W^l$ and $B^l$, respectively. The output processed by each neuron is related to its inputs via a feedforward function followed by a rectified linear unit (ReLU) activation function. Specifically, given the input vector $x \in \mathbb{R}^{N^l-1}$, the feedforward function $f^l : \mathbb{R}^{N^l-1} \rightarrow \mathbb{R}^{N^l}$ for layer $l$ is defined as $f^l(x) = W^l x + b^l$ and the activation function is $\alpha(x) = \max(0, x)$. The NN in Figure 1 is equivalent to the SDT in Figure 2 in the sense that the corresponding input-output characteristic functions $\phi_N(x)$ and $\phi_x(x)$ are identical, as stated by the following proposition.

**Proposition 1:** For the characteristic functions $\phi_N(x)$ and $\phi_x(x)$ associated with the NN and the SDT in Figure 1 and Figure 2, respectively, the following holds:

$$\phi_N(x) = \phi_x(x) \in \begin{cases} 
\{1\} & \text{if } x_0 + x_1 > 2, \\
\{1, 2\} & \text{if } x_0 + x_1 = 2, \\
\{2\} & \text{if } x_0 + x_1 < 2.
\end{cases}$$

**Proof:** From the NN, if $x_0 + x_1 > 2$ holds, then we have $f_3^2(x) = W^2 x + B^2 = x_0 + x_1 - 1 > 0$, hence $\alpha(x_0 + x_1 - 1) = x_0 + x_1 - 1$. Moreover, for the third layer, we obtain $f_3^0 = W^3 x + B^3 = (x - 1, 0)^T$, hence, by function composition, $f_3^0 = f_3^0 \circ \alpha \circ f_3^2(x) = f_3^2(x_0 + x_1 - 1) = (x_0 + x_1 - 2, 0)^T$ and $\phi_N(x) = \arg\max(f_3^0(x)) = 1$. Similarly, when $x_0 + x_1 = 2$ holds, we obtain $f_3^0 \circ \alpha \circ f_3^2(x) = (0, 0)^T$ and conclude $\phi_N(x) \in \{1, 2\}$. Finally, when $x_0 + x_1 < 2$ holds, if $x_0 + x_1 \geq 1$, we obtain $f_3^0 \circ \alpha \circ f_3^2(x) = (x_0 + x_1 - 2, 0)^T$. Otherwise, if $x_0 + x_1 < 1$, we obtain $f_3^0 \circ \alpha \circ f_3^2 = (-1, 0)^T$. In both cases, we have $\phi_N(x) = 2$. In the SDT, decisions are taken based on the probability values computed at each node. If $x_0 + x_1 > 2$ holds, we obtain $p_{v_1}(x) = \sigma(x_0 + x_1 - 1) > 0.5$, hence we proceed to...
node \( v_3 \). By \( p_{v_3}(x) = \sigma(x_0 + x_1 - 2) > 0.5 \), we then reach the leaf node \( v_5 \) via the path \( v_1 \rightarrow v_3 \rightarrow v_5 \). We then obtain \( \phi_3(x) = \arg\max_k(Q_k(x)) = 1 \). When \( x_0 + x_1 = 2 \) holds, then we can follow either path \( v_1 \rightarrow v_3 \rightarrow v_4 \) or \( v_1 \rightarrow v_3 \rightarrow v_5 \), hence \( \phi_3(x) \in \{1, 2\} \). Finally, if \( x_0 + x_1 < 2 \) holds, we can follow either path \( v_1 \rightarrow v_2 \) or \( v_1 \rightarrow v_3 \rightarrow v_4 \), leading both to \( \phi_3(x) = 2 \).

An equivalent SDT can be obtained, for example, by applying the algorithm proposed by Nguyen et al. [30], whose space complexity is exponential in the number of neurons. While results from the properties of hyperplane arrangements [41] suggest that this transformation is not minimal, and space complexity can be significantly reduced by pruning redundant branches, constructing an equivalent SDT for large NNs can still be expensive. This motivates the search for distilled architectures that can be more scalable as well as metrics by which distillation performance can be assessed, as further detailed below.

V. CONTROLLER CHARACTERIZATION METRICS

Given a reference expert controller and a collection of distilled models, it is desirable to identify which of the models represents the “best” approximation to the expert. Much recent work on distillation techniques and their application focuses on controller performance [6], [10], [16]. Still, controller performance alone does not indicate the degree to which the distilled controller “matches” or explains the target DNN policy, particularly in cases where fitted models have lower performance than the expert [10]. Another natural evaluation metric for distilled models is fidelity, or the extent to which fitted models match the predictions of the expert [23]. In the following OpenAI gym environment case studies, we evaluate surrogate student models distilled from a reference DQN-trained DNN on the basis of controller performance as well as fidelity.

We consider policy accuracy over a validation set \( V \) taken from a reference controller trajectory dataset. This figure can also be considered as the percent 0-1 loss, which we later report as a percentage

\[
\%\text{ACC}(\pi_E, \pi_S, V) = \frac{|\{s \in V : \pi_S(s) = \pi_E(s)\}|}{|V|},
\]

where \( \pi_E \) is the output policy action of the expert, or reference controller, and \( \pi_S \) is the output policy action of the surrogate or student controller. This notion of fidelity avoids computational issues arising from continuous or very large discrete state spaces, where direct comparison of policies across the state space is not tractable. Finally, we consider the complexity of candidate controllers in terms of the number of terms required to characterize the models.

VI. CASE STUDIES

We consider the following three benchmarks environments from the OpenAI gym package to illustrate our results. In each case study we examine two surrogate architectures: standard decision trees and Gaussian KMs.

A. MountainCar-v0

The MountainCar [28] environment case study was carried out on a 2.7-GHz Intel Core i5 CPU with 8-GB RAM, Tensorflow 1.15.0 for training the DNN, sklearn 0.22.2 for training the KMs, and gym 0.17.3. In particular, we tested the MountainCar-v0 OpenAI Gym environment [34], in which three actions in total are allowed at each step: push left, push right, or do nothing.

1) Expert DQN: Our expert NN consisted of two hidden layers with 24 and 48 units and ReLU activation, and three units with linear activation to account for the three possible actions. The discount factor here, as well in the later case studies, was set to 0.99 for training. This architecture was trained using the DQN algorithm on 400 episodes of MountainCar-v0. Optimization of the DQN beyond an acceptable performance level was not the focus, so the model architecture and hyperparameters were not tuned beyond the parameters just stated. The training code can be found as part of a repository [37]. The output reference model has a total of 1419 trainable parameters. The DQN learned the optimal control policy successfully, i.e., it reached the goal in each of 100 test episodes, with a mean cumulative reward and 95% confidence interval of \(-130.9 \pm 4.79\) units.

Decision trees and Gaussian KMs were trained on trajectories generated by the trained DQN. Specifically, a set of 1500 trajectories were generated, giving 224758 labeled data points in total. From this set, experimentation showed that selecting just 250 points provided a sufficiently large data set to match the DNN performance with both decision trees and KMs. This subset was partitioned into a 90/10 split of training and validation datasets.

2) Decision Trees: Decision trees of depths 2 through 9 were trained using the sklearn DecisionTreeClassifier module. As Figure 3 shows, validation set accuracy increases monotonically with tree depth, in line with the roughly monotonic increase in average performance in Figure 4.

3) Kernel Machines: All combinations of kernel parameter \( \gamma \in \{0.1, 1, 10.0\} \) and misclassification penalty parameter \( C \in \{0.01, 1, 100.0\} \) were tested in training the Gaussian KMs. The sklearn StandardScaler module was used to transform the training dataset to have zero mean and unit variance before training the KMs. The fitted scaler was then used to transform observations from the validation set and from the environment when testing the KM performance.

Figures 3 and 4 include the validation set accuracy and performance results (aggregated over 100 episodes), respectively, for the Gaussian KMs, where in both plots we order the KMs by mean performance. As can be seen in Figure 3, model accuracy increases monotonically in \( \gamma \) for higher values of \( C \), with every model using \( C \in \{1, 100.0\} \) achieving at least 80% accuracy. On the other hand, Figure 4 shows that the average performance over 100 episodes of two of the KMs falls within or above the 95% confidence intervals of the DQN-trained DNN controller.

4) Discussion: For the MountainCar-v0 environment, both decision trees and Gaussian KMs accurately matched the DNN policy on a sample validation set, as well as...
meet or exceed DNN performance levels, though the KM performance is somewhat sensitive to hyperparameter choice.

Taking the smallest, high performing KM, with hyperparameter values \((C, \gamma) = (1.0, 10.0)\), the model uses 99 two-entry support vectors. Thus, together with the support vectors, this KM is described by \((3 - 1) \cdot 99 = 198\) weight coefficients, giving 396 terms to store overall. On the other hand, above depth 5, the decision trees essentially match or exceed DQN performance. The depth 5 tree consists of 27 nodes overall, including 14 leaf nodes.

1) Expert DQN: Our NN included 4 dense layers, giving 38,531 trainable parameters overall. We tested the DQN controller over 100 episodes, and observed that the controller succeeded in obtaining the maximum reward of 200 in each episode. KMs were trained on a subset of labeled data generated by the expert DQN over 1500 CartPole-v0 episodes totalling 300,000 data points. Again, a small set of just 250 data points were randomly selected and partitioned into a 90/10 split of training and validation datasets.

2) Decision Trees: Decision trees of the same depth range as the MountainCar-v0 study were trained. Again, validation set accuracy roughly increases with tree depth, as shown in Figure 5. However, as Figure 6 illustrates, this trend does not necessarily map to performance for the CartPole-v0 environment.

3) Kernel Machines: Gaussian KMs were trained over the same set of hyperparameters as in the MountainCar-v0 study, with the inputs similarly standardized. Figures 5 and 6 include the validation set accuracy and performance results (averaged over 100 episodes), respectively, for the Gaussian KMs. Again, we sort KMs by mean performance in both plots. As in the MountainCar-v0 case, the highest performing KMs also feature the highest validation accuracy.

4) Discussion: The KMs matching the DQN performance can be described with 77 four-entry support vectors, together with 77 weights (as learning from labeled CartPole-v0 data is an instance of binary classification), giving 385 terms to store in total. Once again, only coarse hyperparameter tuning is required in order to produce Gaussian KMs with comparable performance to the DNN.

In the case of decision trees, a depth 2 tree with 4 output nodes is capable of matching the DQN performance, reflecting the fact that the decision to push left or right basically hinges on the pole velocity, giving an optimal bang-bang type policy. While decision trees are comparable in performance to KMs for the two simple benchmarks presented thus far, given a very small set of expert demonstrations, the following case study demonstrates that for much higher dimensional state space environments, effective KM surrogate controllers are easier to design.

B. CartPole-v0

We now detail the CartPole environment [4] case study, in particular the OpenAI Gym environment Cartpole-v0 [33], in which actions are restricted to applying unit force to the cart in the left or right direction. The computing infrastructure used for this case study is the same as for the MountainCar-v0 case study, but we used Tensorflow 2.1.0 and Stable-Baselines 2.10.1 [12] for the DNN training, sklearn 0.22.1 for training the KMs, and gym 0.18.0.

1) Expert DQN: Our NN included 4 dense layers, giving 38,531 trainable parameters overall. We tested the DQN controller over 100 episodes, and observed that the controller succeeded in obtaining the maximum reward of 200 in each episode. KMs were trained on a subset of labeled data generated by the expert DQN over 1500 CartPole-v0 episodes totalling 300,000 data points. Again, a small set of just 250 data points were randomly selected and partitioned into a 90/10 split of training and validation datasets.

2) Decision Trees: Decision trees of the same depth range as the MountainCar-v0 study were trained. Again, validation set accuracy roughly increases with tree depth, as shown in Figure 5. However, as Figure 6 illustrates, this trend does not necessarily map to performance for the CartPole-v0 environment.

3) Kernel Machines: Gaussian KMs were trained over the same set of hyperparameters as in the MountainCar-v0 case study, with the inputs similarly standardized. Figures 5 and 6 include the validation set accuracy and performance results (averaged over 100 episodes), respectively, for the Gaussian KMs. Again, we sort KMs by mean performance in both plots. As in the MountainCar-v0 case, the highest performing KMs also feature the highest validation accuracy.

4) Discussion: The KMs matching the DQN performance can be described with 77 four-entry support vectors, together with 77 weights (as learning from labeled CartPole-v0 data is an instance of binary classification), giving 385 terms to store in total. Once again, only coarse hyperparameter tuning is required in order to produce Gaussian KMs with comparable performance to the DNN.

In the case of decision trees, a depth 2 tree with 4 output nodes is capable of matching the DQN performance, reflecting the fact that the decision to push left or right basically hinges on the pole velocity, giving an optimal bang-bang type policy. While decision trees are comparable in performance to KMs for the two simple benchmarks presented thus far, given a very small set of expert demonstrations, the following case study demonstrates that for much higher dimensional state space environments, effective KM surrogate controllers are easier to design.
Each DNN was trained using the DQN algorithm, using the default parameters found in a reference repository [1]. In each case targeting a single track, DQN learned a successful policy, i.e., a policy navigating the car through a full loop with score of at least 850. For the three-track case, the DQN learned a policy which was successful across all tracks.

Gaussian KMs were trained on labeled data in each case. For the individual track experiments, the training data consisted of 10 episodes of frame/action trajectories, giving 2310 samples for the elliptical track, 2500 samples for the square track, and 3320 samples for the zigzag track. For the three-track experiment, the training data consisted of 4 episodes of frame/action trajectories for each track, giving 4068 samples. In all experiments, we randomly partitioned the trajectory data into a 90/10 split of training and validation datasets.

3) Decision Trees: Decision trees of depths 2 through 20 were trained and tested. However, due to poor performance results for smaller trees, we focus here on depths 8 through 20. For the three track set, Figure 8 shows that accuracy improves monotonically with increasing tree depth, reaching above 99% accuracy by depth 20 for the full track set. In Figure 9, it can be seen that decision trees of depth 16 and above match the DQN performance, while for the three track set, depth 20 is required to match the DQN performance.

4) Kernel Machines: Gaussian KMs were trained and tested with all combinations of kernel parameter \( \gamma \in \{0.1, 1, 10\} \) and misclassification penalty parameter \( C \in \{0.1, 1, 10\} \). As in the MountainCar-v0 case study, the \texttt{sklearn} StandardScaler module was used to standardize the training dataset, and then transform the input observations when testing the KMs.

Figures 10 and 11 plot the validation set accuracy and performance results, respectively for the Gaussian KMs. In all experiments, including the three track experiment, KMs using misclassification penalty parameter \( C = 1 \) or \( C = 10 \) achieve perfect accuracy, and match the DQN performance, regardless of the value of \( \gamma \). Note that, unlike in the MountainCar-v0 and Cartpole-v0 environments, there is no randomness in initial position on a given track, and in the three track setting, tracks appear in a deterministic order from episode to episode. Therefore, as all controllers considered are deterministic, the performance values are exact, rather than mean values with confidence intervals.

5) Discussion: Overall, both decision trees and KMs are capable of matching DQN performance, and accurately mimicking the DQN policy on the respective validation datasets. While in the case of HDTs the results change considerably with tree depth, the KM results are relatively insensitive to parameter choices. Another notable characteristic of the KMs is that the best performing machines are the simplest in terms of number of support vectors used in the online decision function. Taking the three track experiment as an example, KMs with \( C = 0.1 \) use 99% of the training set input points, while those with \( C = 1 \) and \( C = 10 \) use 70% and 28% of the input points, respectively. Thus, the smallest KM uses 1139 support vectors, i.e., images, meaning its decision rule includes \((12 - 1) \cdot 1139 = 12529\) weights. In total, the KM is described by 10,509,553 terms, the bulk of which...
are accounted for by the stored support vectors. Again, for lower values of misclassification penalty parameter $C$, more training samples can be identified as support vectors than in the cases of higher $C$ values. However, together, Figures 10 and 11 show that high accuracy is crucial to the performance of the KM controllers.

![Decision Tree Validation Set Accuracy](image_url)

**Fig. 8.** CarRacing-v0 decision tree validation set accuracy.

![Decision Tree Performance](image_url)

**Fig. 9.** CarRacing-v0 decision tree performance.

**VII. CONCLUSION AND FUTURE WORK**

In this paper, we first showed via an example that a NN-controller can be equivalently converted into an SDT-controller, thus allowing for the possibility of verifying the properties of the latter to ensure those of the former. We then empirically showed that decision trees and SVMs-based controllers can be constructed using standard IL techniques that approximate the performance of state-of-the-art deep RL-based controllers for three benchmark OpenAI gym environments that range from fairly simple to reasonably complex. While the distilled non-DNN controllers may sacrifice a bit in performance over the DNN controllers, they may be more amenable to formal analysis of their properties, such as safety. They are also more compact models (needing fewer parameters to be specified) and tend to be more interpretable. Our observations can then lead to a control design paradigm for certain applications, based on using deep RL methods to design controllers that perform reasonably well empirically, and then distilling those controllers to non-DNN architectures. If the required properties of the distilled controllers can be verified, they can then be deployed in real systems with only a small sacrifice in performance as compared to the deep RL-based controllers.

We also introduced an evaluation framework and metrics to assess the quality of distilled controllers in RL settings against a pair of design objectives. Specifically, we consider policy match accuracy and mean performance with confidence bounds, to guide designers and researchers toward assessing tradeoffs between performance, fidelity, and complexity across conventional decision trees and Gaussian kernel SVM surrogates.

Future directions for our work include incorporating more sophisticated IL algorithms such as DAGGER [39] into the training of our alternative architectures, and developing metrics which focus more on the closed-loop behavior of controllers in a given environment.

While the MountainCar-v0 environment studied here is fairly simple, CarRacing-v0 in some sense captures the fundamental issues arising when attempting to derive autonomous driving policies directly from input sensor data. As a longer term goal for this work, we hope to address...
more realistic, 3D driving simulators such as CARLA [13] as well as problems involving real-world data sets from applications such as healthcare, including, for example, the cervical cancer risk factor data set [14].

REFERENCES

[16] OpenAI. Cartpole-v0. 1 2022.
[17] OpenAI. Mountaincar-v0. 1 2022.