Neural Networks 151 (2022) 34–47

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

MoËT: Mixture of Expert Trees and its application to verifiable reinforcement learning

Marko Vasić [a,*], Andrija Petrović [b], Kaiyuan Wang [c], Mladen Nikolić [d], Rishabh Singh [e], Sarfraz Khurshid [a]

[a] The University of Texas at Austin, USA
[b] Singidunum University, Serbia
[c] Google, USA
[d] University of Belgrade, Serbia
[e] Google Brain, USA

ARTICLE INFO

Article history:
Received 22 October 2021
Received in revised form 18 February 2022
Accepted 14 March 2022
Available online 23 March 2022

Keywords:
Verification
Deep learning
Reinforcement learning
Mixture of Experts
Explainability

ABSTRACT

Rapid advancements in deep learning have led to many recent breakthroughs. While deep learning models achieve superior performance, often statistically better than humans, their adoption into safety-critical settings, such as healthcare or self-driving cars is hindered by their inability to provide safety guarantees or to expose the inner workings of the model in a human understandable form. We present MoËT, a novel model based on Mixture of Experts, consisting of decision tree experts and a generalized linear model gating function. Thanks to such gating function the model is more expressive than the standard decision tree. To support non-differentiable decision trees as experts, we formulate a novel training procedure. In addition, we introduce a hard thresholding version, MoËT$_h$, in which predictions are made solely by a single expert chosen via the gating function. Thanks to that property, MoËT$_h$ allows each prediction to be easily decomposed into a set of logical rules in a form which can be easily verified. While MoËT is a general use model, we illustrate its power in the reinforcement learning setting. By training MoËT models using an imitation learning procedure on deep RL agents we outperform the previous state-of-the-art technique based on decision trees while preserving the verifiability of the models. Moreover, we show that MoËT can also be used in real-world supervised problems on which it outperforms other verifiable machine learning models.

1. Introduction

Deep learning has achieved many recent breakthroughs, in challenging domains such as Go (Silver et al., 2016), and healthcare (Esteva et al., 2019; Miotto, Wang, Wang, Jiang, & Dudley, 2018) to name a few. Encoding state representation via deep neural networks allows DRL agents to achieve superior performance. Also it enables development of performant radiology models (Cheng et al., 2016; Cicero et al., 2017; Kooi et al., 2017). However, the models learned do not provide safety guarantees and are hard to analyze, which hinders their use in safety-critical applications.

An effective recent approach, called Viper, follows the DAgger imitation learning procedure (Ross, Gordon, & Bagnell, 2011) to create a decision tree model mimicking a DRL agent (Bastani, Pu, & Solar-Lezama, 2018). The key advantage of such decision tree models is that they are amenable to verification. Moreover, they are shown to perform well on environments such as Pong. However, decision trees are limited to axis perpendicular decision boundaries, which can adversely impact the performance. In this paper, we alleviate this issue by proposing a model with less restrictions on the geometry of decision boundaries.

We present MoËT (Mixture of Expert Trees), a technique based on Mixture of Experts (MoE) (Jacobs, Jordan, Nowlan, Hinton, et al., 1991; Jordan & Xu, 1995; Yuksel, Wilson, & Gader, 2012). MoËT consists of DT experts and a gating function that determines the weights with which experts are used. Standard MoE models can typically use any expert as long as it is a differentiable function of model parameters. In this paper we tackle the problem of using non-differentiable decision trees in MoE context, as a means of obtaining verifiable DRL agents. Similar to MoE training by Expectation–Maximization (EM) algorithm, we first observe that MoËT can be trained by interchangeably optimizing the weighted log likelihood for experts (independently from one another) and optimizing the gating function with respect to the obtained experts. Based on that, we propose a procedure for DT learning in the specific context of MOE. To the best of our

* Corresponding author.
E-mail address: vasic@utexas.edu (M. Vasić).

https://doi.org/10.1016/j.neunet.2022.03.022
0893-6080/

knowledge we are first to combine standard non-differentiable DT experts with MoE approach.

For a gating function, we use a simple generalized linear model with softmax function, which provides a distribution over experts. While decision boundaries of DTs are axis-perpendicular, the softmax gating function supports boundaries with hyperplanes of arbitrary orientations, thus improving expressiveness. We also consider a variant of MoËT model that uses hard thresholding (MoËT$_h$) which selects just one most likely expert tree. Since MoE training algorithm tends to assign a region of space to a single expert ($P(e|r) \approx 1$) anyway, this variant does not suffer in performance, as we empirically demonstrate. Benefits of MoËT$_h$ compared to the soft version of MoËT are that it (a) allows for decomposing a decision into a set of logical rules, thus providing means for interpreting the model decisions, and (b) allows translation to satisfiability modulo theories (SMT)[1] formulas (Biere, Heule, van Maaren, & Walsh, 2009), thus providing rich opportunities for formal verification using off the shelf SMT solvers,[2] as we demonstrate in the paper.

To employ MoËT in DRL setting we use the DAgger imitation learning procedure to mimic DRL agents. We evaluate our technique on six different environments: CartPole, Pong, Acrobot, Mountaincar, Lunarlander and Pendulum. We show that MoËT achieves better rewards and lower misprediction rates than Viper. Finally, we demonstrate how a MoËT policy for CartPole can be translated into an SMT formula to verify its properties using the Z3 theorem prover (De Moura & Bjørner, 2008). In addition we showed that MoËT can also be used in real-world supervised machine learning problems. We demonstrated that compared to the other verifiable machine learning models (logistic regression, decision trees and support vector classifiers with linear kernels) MoËT achieved much better results. By improving reliability of AI systems and to a degree improving their interpretability, our work aims at positive societal impact.

In summary, this paper makes the following key contributions:

1. We propose MoËT, a technique based on MoE with decision tree experts, and present a learning algorithm to train MoËT models.
2. We create MoËT$_h$, MoËT version with hard thresholding and softmax gating function which can be translated to an SMT formula amenable for verification and is not hard to interpret in case of small models.
3. We apply MoËT models in the RL setting, evaluate it on different environments and show that they lead to more performant models compared to Viper decision trees.
4. We apply MoËT models in real-world supervised problems and show that MoËT achieved better results compared to the others verifiable machine learning models.

The remainder of the paper is structured as follows. In Section 2 the related work is reviewed. Motivating example to showcase some of the key difference between Viper and MoËT is presented in Section 3, whereas background methodology is presented in 4. Explanation of MoËT model is given in Section 5. Experimental setup and results obtained on different RL environments and supervised datasets are presented in Section 6. The conclusions are drawn in Section 7.

[1] Very roughly, SMT is the problem of determining whether a mathematical formula is satisfiable, and it generalizes the Boolean satisfiability problem (SAT) to more complex formulas.

[2] SMT solvers are tools designed to solve SMT problems.

2. Related work

Verifiable Machine Learning: RL algorithms are notoriously hard to debug and verify (Amir, Schapira, & Katz, 2021; Van Wesel & Goodloe, 2017). A number of techniques has been proposed for enabling verification in RL setting (Kazak, Barrett, Katz, & Schapira, 2019; Li, Serlin, Yang, & Belta, 2019; Verma, Le, Yue, & Chaudhuri, 2019; Zhu, Xiong, Magill, & Jagannathan, 2019). One existing approach synthesizes a program that approximates an RL policy (Zhu et al., 2019). The program acts as a shield, and their technique coordinates between using the shield program and original policy, which in combination provide safety guarantees. Instead of using a programmatic policy as a shield, another approach (Verma et al., 2019) creates a programmatic policy that can replace neural network policy altogether. Niu, Wu, Tang, Ma, and Chen (2020) provide a general framework that leverages the success of verifiable and safe model-free RL in learning high performance controllers. Another system for verification of deep RL agents is presented in Kazak et al. (2019). A hybrid RL agent framework that produces high-level autonomous verifiable behavior for unmanned vehicles is introduced in Wang and Pandit (2019). An abstraction approach, based on interval Markov decision processes, that yields probabilistic guarantees on accuracy of policy's execution, and presents techniques to build and solve different kind of control problems using abstract interpretation, mixed-integer linear programming, entropy-based refinement, and probabilistic model checking is presented in Bacci and Parker (2022).

Compared to the other approaches, in this paper we propose a pure machine learning technique that is verifiable and applicable even outside of the RL setting. There has also been recent work on verification of random forests and tree ensembles (Törnblom & Nadjm-Tehrani, 2018, 2020). Such approaches might be useful in our future work to extend verification from MoËT$_h$ to general MoËT models (which we describe later).

Explainable Machine Learning: There has been a lot of recent interest in explaining decisions of black-box models (Doshi-Velez & Kim, 2017; Guidotti et al., 2018; Roscher, Bohn, Duarte, & Garcke, 2020). Nowadays, a large set of explainable RL literature is emerging, intended to provide ethical, responsible and trustable algorithms for explaining model outputs of DRL agents (Heuillet, Couthouis, & Díaz-Rodríguez, 2021; Puiutta & Veith, 2020; Wells & Bednarz, 2021). Shi, Li, Li, Pan, and Liu (2021) proposed XPM—an explainable RL framework for portfolio management optimization that is based on application of class activation mappings for output explanation. Similarly, Ayala, Cruz, Fernandes, and Dazeley (2021) proposed the introspection-based method for transforming Q-values into probabilities of success, used as the base to explain the agent's decision-making process. Besides of the explainable RL algorithms, the two most well known algorithms that are commonly used for deep learning models interpretation are LIME (Ribeiro, Singh, & Guestrin, 2016) and LORE (Guidotti et al., 2018). LIME and LORE explain behavior of a black-box model locally, around an input of interest, by sampling the black-box model around the neighborhood of the input, and training a local DT (or a linear model) over the sampled points.

Another view at MoËT is that it explains behavior of a deep RL agent. MoËT combines local trees into a global policy by combining local decision trees via a gating function. Inspection of the trees and the gating might shed light on the agent's decision making. However, we do not focus on this aspect in this paper.

Tree-Structured Models: Tree-Structured models are very attractive type of machine learning algorithms due to low complexity and interpretability (Kotsiantis, 2013; Niuniu & Yuxun, 2010). Irsoy, Yıldız, and Alpaydın (2012) propose a decision tree model with soft decisions at internal nodes where children are chosen

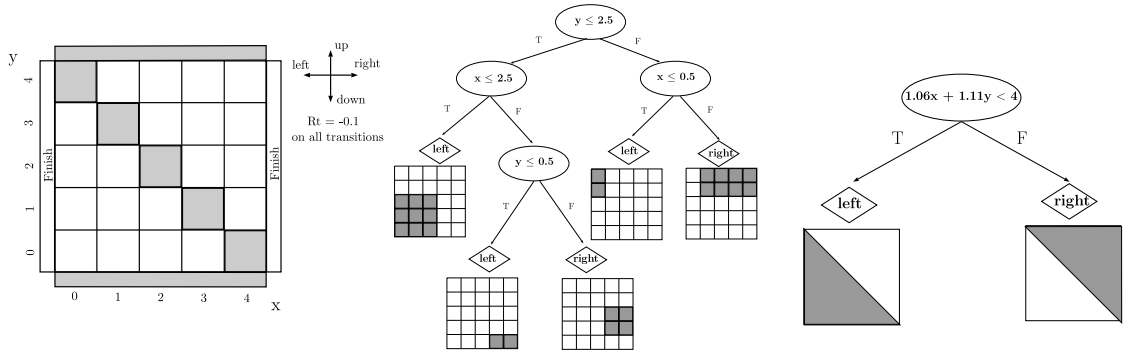

Fig. 1. An illustratory Gridworld environment (left), a Viper policy learned for the environment (middle), and a MoËT policy learned for the environment (right).

with probabilities given by a sigmoid gating function. However, this reduces the tree's interpretability. Binary tree-structured hierarchical routing mixture of experts (HRME) model, which has classifiers as non-leaf node experts and simple regression models as leaf node experts, was proposed in Zhao, Gao, Memon, Raj, and Singh (2019). Hester and Stone (2013) use random forests in RL setting to build a model of environment from which policy is inferred.

The form of our model can be related to these models, but it is designed with verifiability in mind and we also propose a novel training procedure suited to that specific model.

Knowledge Distillation and Model Compression: We rely on ideas already explored in fields of model compression (Buciluǎ, Caruana, & Niculescu-Mizil, 2006) and knowledge distillation (Gou, Yu, Maybank, & Tao, 2021; Hinton, Vinyals, & Dean, 2015; Wang, Wei, & Wu, 2021). The idea is to use a complex well performing model to facilitate training of a simpler model which might have some other desirable properties (e.g., verifiability and interpretability). Such practices have been applied to approximate decision tree ensemble by a single tree (Breiman & Shang, 1996). In contrast, we approximate a neural network. Similarly, a neural network can be used to train another neural network (Furlanello, Lipton, Tschannen, Itti, & Anandkumar, 2018), but neural networks are hard to interpret and even harder to formally verify. Such practices have also been applied in the field of reinforcement learning in knowledge and policy distillation (Gao, Xu, Ding, & Wang, 2021; Koul, Fern, & Greydanus, 2019; Rusu et al., 2016; Tsantekidis, Passalis, & Tefas, 2021; Zhang et al., 2019), which are similar in spirit to our work, and imitation learning (Abbeel & Ng, 2004; Bastani et al., 2018; Ross et al., 2011; Schaal, 1999), which provide a foundation for our work.

3. Motivating example: gridworld

We now present a simple motivating example to showcase some of the key differences between Viper and MoËT approaches. Consider the $N \times N$ Gridworld problem shown in Fig. 1 (for $N = 5$). The agent is placed at a random position in a grid (except the walls denoted by filled rectangles) and should find its way out. To move through the grid the agent can choose to go up, left, right or down at each time step. If it hits the wall (gray cell) it stays in the same position (state). State is represented using two integer values (x, y coordinates) which range from (0, 0)—bottom left to $(N-1, N-1)$—top right. The grid can be escaped through either left doors (left of the first column), or right doors (right of the last column). A negative reward of -0.1 is received for each agent action (negative reward encourages the agent to find the exit as fast as possible). An episode finishes as soon as an exit is reached or if 100 steps are made whichever comes first.

The optimal policy (π_*) for this problem consists of taking the left (right resp.) action for each state below (above resp.) the diagonal. We used π_* as a teacher and imitation learning approach of Viper to train an interpretable DT policy that mimics π_*. The resulting DT policy is shown in Fig. 1. The DT partitions the state space (grid) using lines perpendicular to x and y axes, until it separates all states above diagonal from those below. This results in a DT of depth 3 with 9 nodes. On the other hand, the policy learned by MoËT is shown in Fig. 1. The MoËT model with 2 experts learns to partition the space using the line defined by a linear function $1.06x + 1.11y = 4$ (roughly the diagonal of the grid). Points on the different sides of the line correspond to two different experts which are themselves DTs of depth 0 always choosing to go left (below) or right (above).

Table 1
Size comparison of MoËT and Viper DT policies on the Gridworld problem (Fig. 1), for different sizes of the square board ($N \times N$). The left side of the table presents the depths of obtained models (that perfectly mimic optimal policy) for MoËT and for Viper (DTs), while the right side presents the number of nodes in these models. Both the depth and the number of nodes show that by increasing size of the grid (N) size of MoËT models stays constant, while Viper (DT) models grow in size.

N	Depth		Nodes	
	MoËT	Viper DT	MoËT	Viper DT
5	1	3	3	9
6	1	4	3	11
7	1	4	3	13
8	1	4	3	15
9	1	4	3	17
10	1	5	3	21

We notice that DT policy needs much larger depth to represent π_* while MoËT can represent it as only one decision step. Furthermore, with increasing N (size of the grid), complexity of DT grows, while MoËT complexity stays the same; we empirically confirm this as follows. For Gridworld sizes $N = 5, 6, 7, 8, 9, 10$, the depths of obtained DTs are 3, 4, 4, 4, 4, 5 and the numbers of their nodes are 9, 11, 13, 15, 17, 21 respectively. In contrast, MoËT models of the same complexity and structure as the one shown in Fig. 1 are learned for all values of N. We present these results in Table 1 for better readability (all policies learned are equivalent to π_*).

4. Background

In this section we provide description of two relevant methods we build upon: (1) Viper, an approach for interpretable imitation learning, and (2) MoE learning framework.

Viper. Viper algorithm (included in Appendix.) is an instance of DAGGER imitation learning approach, adapted to prioritize critical states based on Q-values. Inputs to the Viper training algorithm are (1) environment e which is an finite horizon (T-step) Markov Decision Process (MDP) (S, A, P, R) with states S, actions

A, transition probabilities $P : S \times A \times S \to [0, 1]$, and rewards $R : S \to \mathbb{R}$; (2) teacher policy $\pi_t : S \to A$; (3) its Q-function $Q^{\pi_t} : S \times A \to \mathbb{R}$ and (4) number of training iterations N. Distribution of states after T steps in environment e using a policy π is $d^{(\pi)}(e)$ (assuming randomly chosen initial state). Viper uses the teacher as an oracle to label the data (states with actions). It initially uses teacher policy to sample trajectories (states) to train a student (DT) policy. It then uses the student policy to generate more trajectories. Viper samples training points from the collected dataset D giving priority to states s having higher importance $I(s)$, where $I(s) = \max_{a \in A} Q^{\pi_t}(s, a) - \min_{a \in A} Q^{\pi_t}(s, a)$. This sampling of states leads to faster learning and shallower DTs. The process of sampling trajectories and training students is repeated for number of iterations N, and the best student policy is chosen using reward as the criterion.

Mixture of Experts. MoE is an ensemble model (Jacobs et al., 1991; Jordan & Xu, 1995; Yuksel et al., 2012) that consists of expert networks and a gating function. Gating function divides the input (feature) space into regions for which different experts are specialized and responsible. MoE is flexible with respect to the choice of expert models as long as they are differentiable functions of model parameters (which is not the case for DTs).

In MoE framework, probability of outputting $\mathbf{y} \in \mathbb{R}^m$ given an input $\mathbf{x} \in \mathbb{R}^n$ is given by:

$$P(\mathbf{y}|\mathbf{x}, \theta) = \sum_{i=1}^{E} P(i|\mathbf{x}, \theta_g) P(\mathbf{y}|\mathbf{x}, \theta_i) = \sum_{i=1}^{E} g_i(\mathbf{x}, \theta_g) P(\mathbf{y}|\mathbf{x}, \theta_i) \tag{1}$$

where E is the number of experts, $g_i(\mathbf{x}, \theta_g)$ is the probability of choosing the expert i (given input $\mathbf{x}$), $P(\mathbf{y}|\mathbf{x}, \theta_i)$ is the probability of expert i producing output $\mathbf{y}$ (given input $\mathbf{x}$). Learnable parameters are $\theta = (\theta_g, \theta_e)$, where θ_g are parameters of the gating function and $\theta_e = (\theta_1, \theta_2, \ldots, \theta_E)$ are parameters of the experts. Gating function can be modeled using a softmax function over a set of linear models. Let θ_g consist of parameter vectors $(\theta_{g1}, \ldots, \theta_{gE})$, then the gating function can be defined as $g_i(\mathbf{x}, \theta_g) = \exp(\theta_{gi}^T \mathbf{x}) / \sum_{j=1}^{E} \exp(\theta_{gj}^T \mathbf{x})$.

In the case of classification, an expert i outputs a vector $\mathbf{y}_i$ of length C, where C is the number of classes. Expert i associates a probability to each output class c (given by $\mathbf{y}_{ic}$) using the gating function. Final probability of a class c is a gate weighted sum of $\mathbf{y}_{ic}$ for all experts $i \in 1, 2, \ldots, E$. This creates a probability vector $\mathbf{y} = (y_1, y_2, \ldots, y_C)$, and the output of MoE is $\arg\max_i \mathbf{y}_i$.

MoE is commonly trained using an EM algorithm, where instead of direct optimization of the likelihood one performs optimization of an auxiliary function $\hat{L}$ defined in a following way. Let z denote the expert chosen for instance $\mathbf{x}$. Then joint likelihood of $\mathbf{x}$ and z can be considered. Since z is not observed in the data, log likelihood of samples $(\mathbf{x}, z, \mathbf{y})$ cannot be computed, but instead expected log likelihood can be considered, where expectation is taken over z. Since the expectation has to rely on some distribution of z, in the iterative process, the distribution with respect to the current estimate of parameters θ is used. More precisely function $\hat{L}$ is defined by Jordan and Xu (1995):

$$\begin{aligned} \hat{L}(\theta, \theta^{(k)}) &= \mathbb{E}_z[\log P(\mathbf{x}, z, \mathbf{y})|\mathbf{x}, \mathbf{y}, \theta^{(k)}] \\ &= \int P(z|\mathbf{x}, \mathbf{y}, \theta^{(k)}) \log P(\mathbf{x}, z, \mathbf{y}) dz \end{aligned} \tag{2}$$

where $\theta^{(k)}$ is the estimate of parameters θ in iteration k. Then, for a specific sample $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, N\}$, the following formula can be derived (Jordan & Xu, 1995):

$$\hat{L}(\theta, \theta^{(k)}) = \sum_{i=1}^{N} \sum_{j=1}^{E} h_{ij}^{(k)} \log g_j(\mathbf{x}_i, \theta_g) + \sum_{i=1}^{N} \sum_{j=1}^{E} h_{ij}^{(k)} \log P(\mathbf{y}_i|\mathbf{x}_i, \theta_j) \tag{3}$$

where it holds

$$h_{ij}^{(k)} = \frac{g_j(\mathbf{x}_i, \theta_g^{(k)}) P(\mathbf{y}_i|\mathbf{x}_i, \theta_j^{(k)})}{\sum_{l=1}^{E} g_l(\mathbf{x}_i, \theta_g^{(k)}) P(\mathbf{y}_i|\mathbf{x}_i, \theta_l^{(k)})} \tag{4}$$

5. Mixture of expert trees

In this section we explain the adaptation of original MoE model to mixture of decision trees, and present both training and inference algorithms.

Considering that coefficients $h_{ij}^{(k)}$ (Eq. (4)) are fixed with respect to θ and that in Eq. (3) the gating part (first double sum) and each expert part depend on disjoint subsets of parameters θ, training can be carried out by interchangeably optimizing the weighted log likelihood for experts (independently from one another) and optimizing the gating function with respect to the obtained experts. The training procedure for MoËT, described by Algorithm 1, is based on this observation. First, the parameters of the gating function are randomly initialized (line 2). Then the experts are trained one by one. Each expert j is trained on a dataset D_w of instances weighted by coefficients $h_{ij}^{(k)}$ (line 5), by applying specific DT learning algorithm (line 6) that we adapted for MoE context (described below). After the experts are trained, an optimization step is performed (line 7) in order to increase the gating part of Eq. (3). At the end, the parameters are returned (line 8).

Our tree learning procedure is as follows. Our technique modifies original MoE algorithm in that it uses DTs as experts. The fundamental difference with respect to traditional model comes from the fact that DTs do not rely on explicit and differentiable loss function which can be trained by gradient descent or Newton's methods. Instead, due to their discrete structure, they rely on a specific greedy training procedure. Therefore, the training of DTs has to be modified in order to take into account the attribution of instances to the experts given by coefficients $h_{ij}^{(k)}$, sometimes called *responsibility* of expert j for instance i. If these responsibilities were hard, meaning that each instance is assigned to strictly one expert, they would result in partitioning the feature space into disjoint regions belonging to different experts. On the other hand, soft responsibilities are fractionally distributing each instance to different experts. The higher the responsibility of an expert j for an instance i, the higher the influence of that instance on that expert's training. In order to formulate this principle, we consider which way the instance influences construction of a tree. First, it affects the impurity measure computed when splitting the nodes and second, it influences probability estimates in the leaves of the tree. We address these two issues next.

A commonly used impurity measure to determine splits in the tree is the Gini index. Let U be a set of indices of instances assigned to the node for which the split is being computed and D_U set of corresponding instances. Let categorical outcomes of y be $1, \ldots, C$, and for $l = 1, \ldots, C$ let denote p_l as a fraction of instances in D_U for which it holds $y = l$. More formally $p_l = \frac{\sum_{i \in U} I[y_i = l]}{|U|}$, where I denotes indicator function of its argument expression and equals 1 if the expression is true. Then the Gini index G of the set D_U is defined by: $G(p_1, \ldots, p_C) = 1 - \sum_{l=1}^{C} p_l^2$. Considering that the assignment of instances to experts are fractional as defined by responsibility coefficients $h_{ij}^{(k)}$ (which are provided to tree fitting function as weights of instances computed in line 5 of the algorithm), this definition has to be modified in that the instances assigned to the node should not be counted, but instead, their weights should be summed. Hence, we propose the following definition:

$$\hat{p}_l = \frac{\sum_{i \in U} I[y_i = l] h_{ij}^{(k)}}{\sum_{i \in U} h_{ij}^{(k)}} \tag{5}$$

Algorithm 1 MoËT training.

procedure MoËT (Data $\{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, N\}$, Epochs N_E, Number of Experts E)
  $\theta_{\mathbf{g}} \leftarrow initialize()$
  for $k \leftarrow 1$ to N_E **do**
    for $j \leftarrow 1$ to E **do**
      $D_w \leftarrow \left\{ \left(\mathbf{x}_i, \mathbf{y}_i, \frac{g_j(\mathbf{x}_i,\theta_g)P(\mathbf{y}_i|\mathbf{x}_i,\theta_j)}{\sum_{e=1}^{E} g_e(\mathbf{x}_i,\theta_g)P(\mathbf{y}_i|\mathbf{x}_i,\theta_e)} \right) \mid i = 1, \ldots, N \right\}$
      $\theta_j \leftarrow fit\_tree(D_w)$
    $\theta_g \leftarrow \theta_g + \lambda \nabla_{\theta'} \sum_{i=1}^{N} \sum_{j=1}^{E} \left[\frac{g_j(\mathbf{x}_i,\theta_g)P(\mathbf{y}_i|\mathbf{x}_i,\theta_j)}{\sum_{e=1}^{E} g_e(\mathbf{x}_i,\theta_g)P(\mathbf{y}_i|\mathbf{x}_i,\theta_e)} \log g_j(\mathbf{x}_i, \theta') \right]$
  return $\theta_g, (\theta_1, \ldots, \theta_E)$

and compute the Gini index for the set D_U as $G(\hat{p}_1, \ldots, \hat{p}_C)$. Similar modification can be performed for other impurity measures (such as entropy) relying on distribution of outcomes of a categorical variable. Note that while the instance assignments to experts are soft, instance assignments to nodes within an expert are hard, meaning sets of instances assigned to different nodes are disjoint. Probability estimate for $\mathbf{y}$ in the leaf node is usually performed by computing fractions of instances belonging to each class. Instead of such estimates, again, we use estimates $\hat{p}_l$ defined by Eq. (5). Hence, the estimates of probabilities $P(\mathbf{y}|\mathbf{x}, \theta_j^{(k)})$ needed by MoE are defined. In Algorithm 1, function *fit_tree* performs decision tree training using the above modifications.

We consider two ways to perform inference with respect to the obtained model. First one which we call MoËT, is performed by maximizing $P(\mathbf{y}|\mathbf{x}, \theta)$ with respect to $\mathbf{y}$ where this probability is defined by Eq. (1). The second way, which we call MoËT$_\text{h}$, performs inference as $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \theta_{\arg\max_j g_j(x,\theta_g)})$, meaning that we only rely on the most probable expert.

Adaptation of MoËT to imitation learning. We integrate MoËT model into imitation learning approach of Viper by substituting training of DT with the MoËT training procedure.

Verifiability by translating MoËT to SMT. We define a translation of MoËT$_\text{h}$ models to SMT formulas, which opens a range of possibilities for validating and interpreting the model using automated reasoning tools. SMT formulas provide a rich means of logical reasoning, where a user can query the solver with questions such as: "What inputs do the two models differ on?", or "What is the closest input to the given input using which model makes a different prediction?", or "Are the two models equivalent?", or "Are the two models equivalent in respect to the output class C?". Answers to such questions can help better understand and compare models in a rigorous way. Also note that the symbolic reasoning of the gating function and decision trees allows construction of SMT formulas that are readily handled by off-the-shelf tools, whereas direct SMT encoding of neural networks do not scale for any reasonably sized network because of the need for non-linear arithmetic reasoning.

We show the translation of MoËT policy to SMT constraints for verifying policy properties. We present an example translation of MoËT policy on CartPole environment with the same property specification that was proposed for verifying Viper policies (Bastani et al., 2018). The goal in CartPole is to keep the pole upright, which can be encoded as a formula:

$$\psi \equiv s_0 \in S_0 \wedge \bigwedge_{t=1}^{\infty} |\phi(f(s_{t-1}, \pi(s_{t-1})))| \leq y_0$$

where s_i represents state after i steps, ϕ is the deviation of pole from the upright position. In order to encode this formula it is necessary to encode the transition function $f(s, a)$ which models environment dynamics: given a state and action it returns the next state of the environment. Also, it is necessary to encode the policy function $\pi(s)$ that for a given state returns action to perform. There are two issues with verifying ψ: (1) infinite time horizon; and (2) the nonlinear transition function f. To solve this problem, Bastani et al. (2018) use a finite time horizon $T_{max} = 10$ and linear approximation of the dynamics. We make the same assumptions.

To encode $\pi(s)$ we need to translate both the gating function and DT experts to logical formulas. Since the gating function in MoËT$_\text{h}$ uses exponential function, it is difficult to encode the function directly in Z3 as SMT solvers do not have efficient decision procedures to solve non-linear arithmetic. The direct encoding of exponentiation therefore leads to prohibitively complex Z3 formulas. We exploit the following simplification of the gating function that is sound when hard prediction is used:

$$\begin{aligned} e = \arg\max_i \left(\frac{\exp(\theta_{gi}^T \mathbf{x})}{\sum_{j=1}^{E} \exp(\theta_{gj}^T \mathbf{x})} \right) &= \arg\max_i (\exp(\theta_{gi}^T \mathbf{x})) \\ &= \arg\max_i (\theta_{gi}^T \mathbf{x}) \end{aligned} \quad (6)$$

First simplification is possible since the denominators of the gating functions are same for all experts, and second is due to the monotonicity of the exponential function. We use the same DT encoding as in Viper. To verify that ψ holds we need to show that $\neg\psi$ is unsatisfiable. In the experimental evaluation we run the verification with our MoËT$_\text{h}$ policies and show that $\neg\psi$ is indeed unsatisfiable.

Expressiveness. DTs make their decisions by partitioning the feature space into regions which have borders perpendicular to coordinate axes. To approximate borders that are not perpendicular to coordinate axes very deep trees are often necessary. MoËT$_\text{h}$ mitigates this shortcoming by exploiting hard softmax partitioning of the feature space using borders which are still hyperplanes, but need not be perpendicular to coordinate axes (see Section 3), which improves the expressiveness.

Interpretability. While we do not focus on interpretability in this work, it is useful to note that MoËT$_\text{h}$ models do exhibit some interpretability properties. A MoËT$_\text{h}$ model is a combination of a linear model and several decision tree models. Only a single DT is used for each prediction (instead of weighted average), which facilitates interpretability. If the models are small (e.g, depth $\leq$ 10) and include small number of features, a person can easily simulate and understand the model. These observations resonate with several points about interpretability made in Lipton (2016)

Limitations. Our work tries to strike a balance between expressiveness, which allows for more performant models, and verifiability, which allows for more reliable models. Therefore, while being more expressive than decision trees, MoËT still has limited expressiveness compared to deep learning models, which is a price paid for easier verifiability.

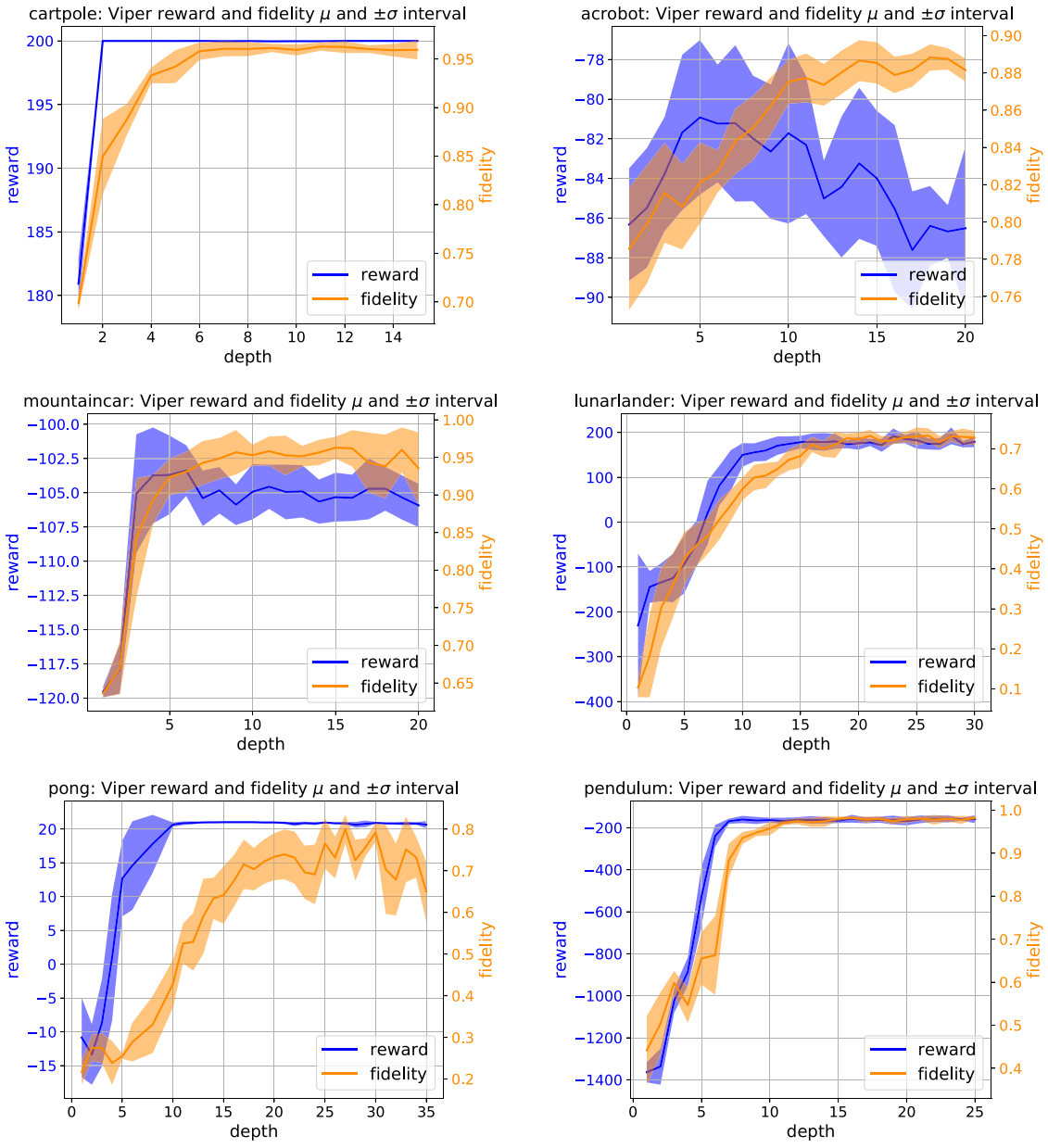

Fig. 2. Performance saturation of Viper. Multiple models are trained for a single maximum depth of Viper decision trees, while maximum depth is incrementally increased, showing the mean value and standard deviation of reward and fidelity with respect to the depth. These results inform when Viper performance saturates, i.e., reaches a point upon which increasing maximum depth is not helpful anymore, we call that point performance saturation depth.

6. Evaluation

We first discuss DRL agents we use as a starting point in the imitation learning. Second, we explore the performance capabilities of Viper by finding decision tree depths at which the performance saturates—cannot be improved by increasing the depth further. Then, after ensuring that we explored the useful space of configurations for Viper, we pick the best performing Viper models and compare them with the best performing MoËT models to quantitatively compare the two. Finally, we re-evaluate performance of the models to evaluate how well they generalize. Also, we verify MoËT_{h} policies on CartPole environment and visually compare the expressiveness of different policies. Eventually, we presented that MoËT can be also successfully applied in real-world supervised learning problems.

DRL agents. We use following OpenAI Gym environments in our evaluation: CartPole, Acrobot, Mountaincar, Lunarlander, Pong and Pendulum (description of the environments is included in the Appendix). For DRL agents, we use a policy gradient model in CartPole, a deep Q-network (DQN) (Mnih et al., 2015) in Pong, and dueling DQN (Wang et al., 2015) in the other environments (training hyperparameters provided in the Appendix). We train MoËT and Viper policies by mimicking the agents. The rewards (total return during an episode) obtained by the DRL agents on CartPole, Acrobot, Mountaincar, Lunarlander, Pong and Pendulum are 200.00, −68.60, −105.27, 190.90, 21.00 and −158.13, respectively. Rewards are averaged across 100 (250 in CartPole) runs (episodes).

Performance saturation of Viper. We first examine performance capabilities of Viper, i.e., answer the question of when the performance saturates, by examining performance of decision trees of gradually increased maximum depth (Fig. 2). For each depth we train multiple Viper models and show performance trends in terms of reward and fidelity. By reward we mean

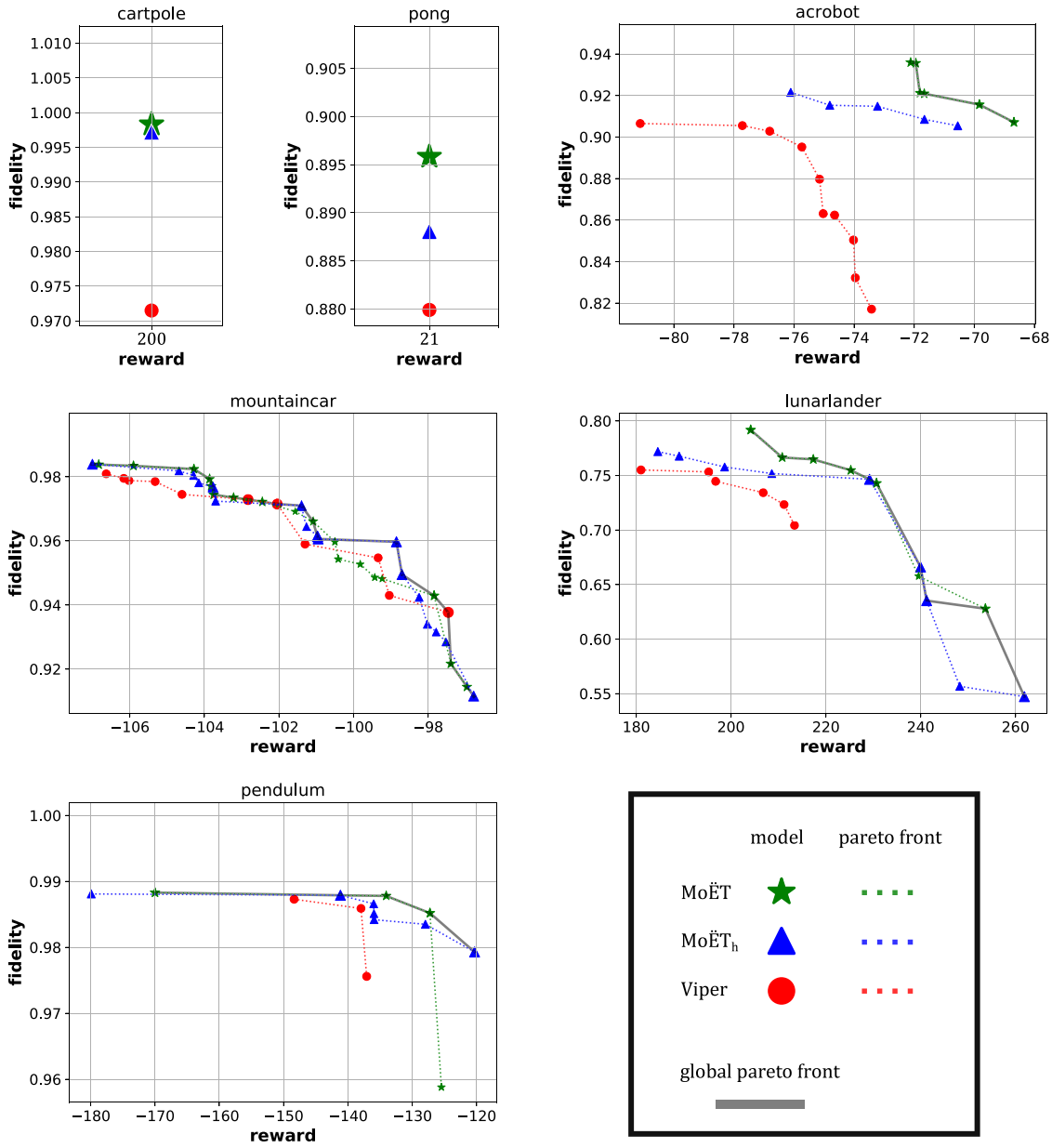

Fig. 3. Best performing Viper, MoËT and MoËT$_h$ models. Pareto fronts (in respect to the reward and fidelity) are identified separately for Viper, MoËT and MoËT$_h$ models. Global Pareto fronts are shown with points connected by a gray solid line.

cumulative reward achieved during an episode, while fidelity represents percent of times a student performs the same action as its teacher (DRL agent). Achieving high reward indicates that a student is performing well, while high fidelity indicates that the student policy is close to the teacher's. We ensure to train at least 5 different Viper models for each depth.[3] Using the performance trend plots we infer when Viper performance saturates, i.e., reaches a depth after which further increasing maximum depth does not help. Performance saturation depths for CartPole, Acrobot, Mountaincar, Lunarlander, Pong and Pendulum are 8, 15, 12, 20, 30 and 20, respectively. Identifying the performance saturation points for Viper is helpful in identifying the overall best performing Viper model, thus giving confidence during comparison with MoËT models that we explored the useful space of Viper configurations.

Best performing Viper, MoËT and MoËT$_h$ models. We next compare Viper, MoËT and MoËT$_h$ models by visualizing their Pareto fronts with respect to the reward and fidelity (Fig. 3). Pareto front of a set of models consists of all models from that set which are not dominated by any other model from the set in terms of reward or fidelity. In other words, every model dominated by another model in terms of both metrics is not considered. From the set of all Viper models trained for different maximum depths (from depth 1 to the saturation depth) we select models on the Pareto front. Similar is done for MoËT and MoËT$_h$ which we trained for different number of experts and expert depths (information about configurations used is provided in the Appendix). A global Pareto front (best models across all architectures) is shown with points connected by a black solid line.

[3] We train at least 5 Viper models for each subject and maximum depth value. Due to the computational limitations actual number of Viper models trained varies across environments: CartPole $\in [35, 70]$, Acrobot $\in [10, 70]$, Mountaincar $\in [10, 70]$, Lunarlander $\in [10, 70]$, Pong $\in [5, 24]$ and Pendulum $\in \{10\}$.

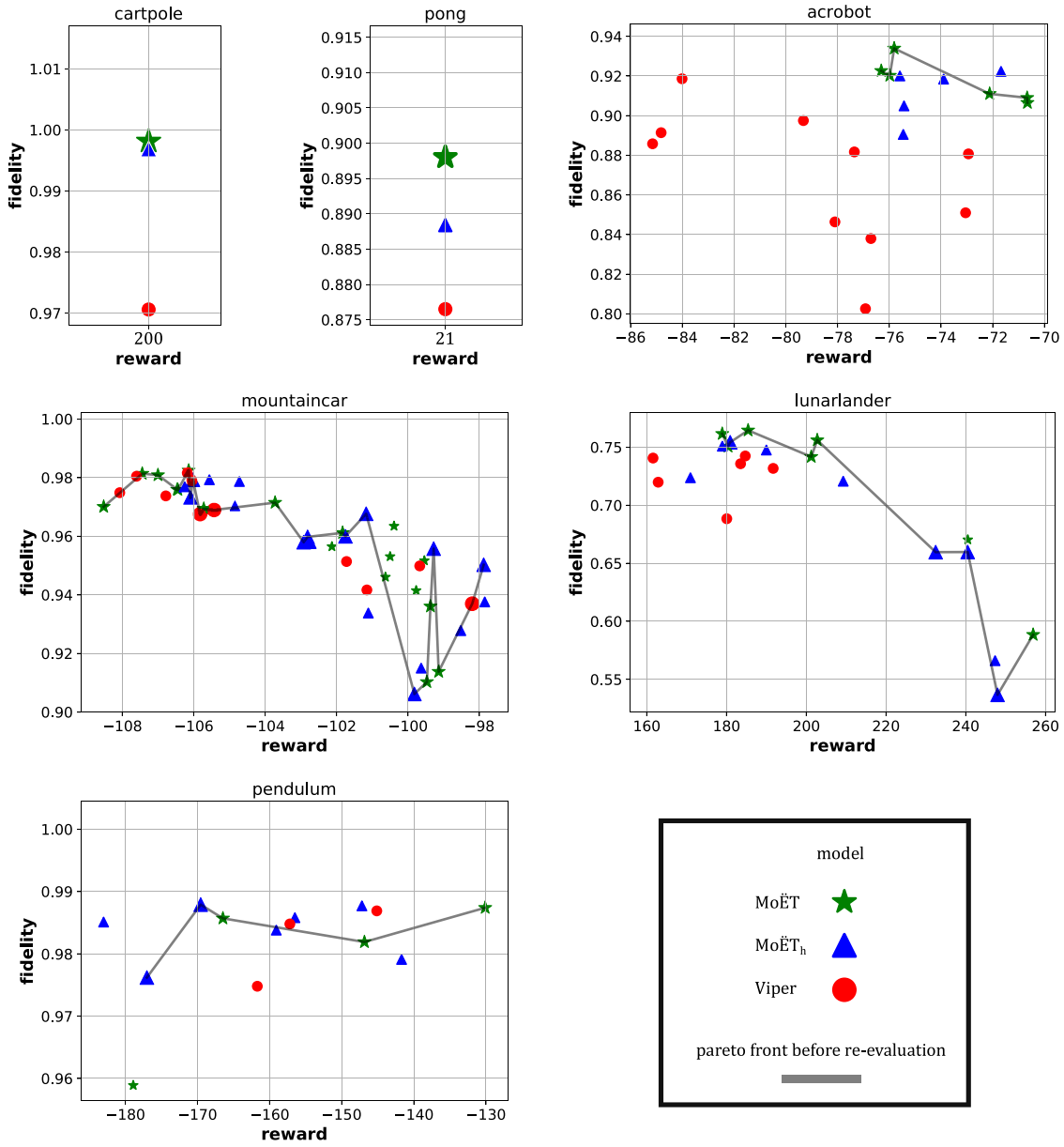

Fig. 4. Performance generalization of models. Models on the Pareto fronts (Fig. 3) are re-evaluated. Black solid line connects models that were on the global Pareto front before re-evaluation.

By inspecting the results we notice that in the case of CartPole, all 3 models achieve maximum reward (200), however fidelity is significantly higher in the case of MoËT and MoËT$_h$ (over 99% compared to 97%). Also, it is interesting to note that both MoËT and MoËT$_h$ models on the Pareto front consist of 2 experts of depth 0, while the Viper model on the Pareto front is a decision tree of depth 6. In the case of Acrobot, we notice that MoËT models dominate MoËT$_h$ and Viper models, and that MoËT$_h$ models dominate Viper models. Thus, both MoËT and MoËT$_h$ models achieve higher reward and fidelity over Viper models. In the case of Mountaincar, the global Pareto front contains some Viper models, but mostly MoËT and MoËT$_h$ dominate. Furthermore, models exhibiting the highest reward as well as fidelity are MoËT and MoËT$_h$ models. In the case of Lunarlander, both MoËT and MoËT$_h$ dominate Viper models. A MoËT$_h$ model achieves the maximum reward of over 260 while a Viper model achieves the maximum reward of around 215. Furthermore, both MoËT and MoËT$_h$ models achieve better fidelity compared to Viper. In the case of Pong, all 3 models achieve maximum reward (21), however fidelity is higher for MoËT and MoËT$_h$. In the case of Pendulum, MoËT and MoËT$_h$ models achieve better maximum reward, while maximum fidelity is about equal for all the models. Note that for a given fidelity score, MoËT and MoËT$_h$ are advantageous to Viper. Scores of the points on the global Pareto front are presented in a tabular form in Appendix E.

Performance generalization of models. In the supervised learning setting, after the best models are selected based on their performance on a validation set, they are re-evaluated on a test set to get a better estimate of their performance on the new data. In RL setting there is no direct analogy to validation and test datasets, but the models can be re-evaluated after the selection is performed. After we identify the best models on the Pareto fronts (Fig. 3), we re-evaluate their performance by running them again through the RL environment. Fig. 4 shows the achieved performance of these models after re-evaluation. In the case of CartPole and Pong performance before and after re-evaluation are very similar. In the case of Acrobot, Mountaincar and Lunarlander, models that were on the global Pareto front are mostly still on

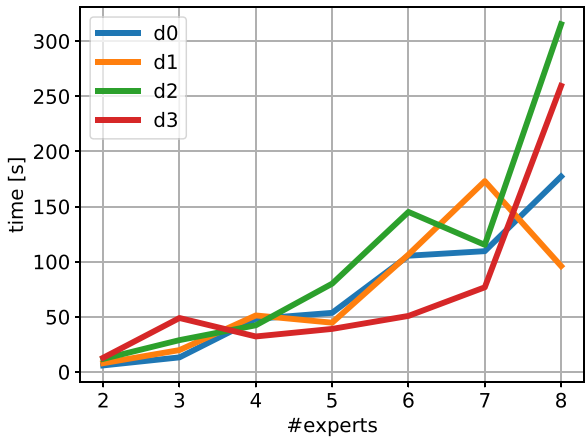

Fig. 5. Verification times.

the global Pareto front in the re-evaluation. Moreover, MoËT and MoËT$_h$ models dominate Viper models in most of the cases. Pendulum environment behaves more stochastically—evaluating policy (done across 100 episodes) can exhibit significantly different reward from evaluation to evaluation, making results more inconclusive. However, all models achieve great fidelity level, and reward that is close to the DRL agent one. Considering high performance, differences in performance between models are minor. Scores of the points that were on the global Pareto front are presented in a tabular form in Appendix E.

Following the previous analysis, we conclude that MoËT and MoËT$_h$ models provide better performance (in terms of reward and fidelity) compared to Viper in most of the cases, demonstrating that MoËT is a valuable technique to be considered when looking for a verifiable RL policy.

Verification. We perform verification of MoËT$_h$ policies obtained in our experiments according to the procedure described in Section 5. All models considered in this experiment successfully pass the verification procedure. To better understand the scalability of our verification procedure, we report the verification times needed to verify policies for different number of experts and expert depths in Fig. 5. The verification times generally increase with the number of experts. MoËT$_h$ policies with 2 experts take from 5.5 s to 11.7s for verification, while the verification times for 8 experts can go up to as much as 336 s. This corresponds to the complexity of the logical formula obtained with an increase in the number of experts. While the effect of expert depths on verification times is visible in a case of few experts, with the increase of experts it is less noticeable, thus indicating that the number of experts has more influence on the verification times than expert depths. We run the verification on Intel i7-7600, 2.80 GHz, 16 GB LPDDR3. We show example SMT formula (of Viper and MoËT$_h$ policies) in Appendix D.

Expressiveness. We provide a simple qualitative comparison of best Viper and MoËT$_h$ policies, by contrasting them to DRL policy on a CartPole environment. Fig. 6 visualizes these policies and demonstrates that MoËT$_h$ policy much more closely resembles the DRL policy thanks to its ability to represent hyperplanes of arbitrary orientation, while DT policy obtained by Viper approximates DRL policy by axis perpendicular hyperplanes. The MoËT$_h$ policy presented is equivalent to the following program: `if` $2.18 * cp + 7.22 * cv + 20.64 * pa + 25.33 * pv > -1$ `then` go right `else` go left, where cp and cv are cart position and velocity, and pv and pa pole angle and its angular velocity.

Supervised learning. We evaluated the performance of MoËT and MoËT$_h$ in the supervised regime on three real-world datasets.

Table 2
For each dataset used in the experimental evaluation we provide its name, the number of instances it contains (Size), numbers of instances per set after splitting the data into training, validation, and testing sets (Split) and total number of features (Features).

Dataset	Size	Split (train/test/val)	Features
Adult income	48,842	34,189 / 6,783 / 6,784	14
German credit	1,000	700 / 150 / 150	10
Fetal health	2,126	1,488 / 319 / 319	21

Table 3
Prediction performance of classifiers—*Fetal health* dataset.

Model/metrics	F1 score	Accuracy
Decision tree	0.852 ± 0.004	0.939 ± 0.004
Lasso logistic regression	0.797 ± 0.000	0.915 ± 0.000
MoËT$_h$	0.880 ± 0.001	0.950 ± 0.001
MoËT	**0.891 ± 0.001**	**0.955 ± 0.001**
Ridge logistic regression	0.739 ± 0.000	0.903 ± 0.000
SVC	0.762 ± 0.000	0.906 ± 0.000

Table 4
Prediction performance of classifiers—*German credit* dataset.

Model/metrics	F1 score	Accuracy
Decision tree	0.759 ± 0.000	0.637 ± 0.000
Lasso logistic regression	0.797 ± 0.000	0.667 ± 0.000
MoËT$_h$	0.759 ± 0.003	0.638 ± 0.004
MoËT	**0.808 ± 0.003**	0.687 ± 0.004
Ridge logistic regression	0.792 ± 0.000	0.660 ± 0.000
SVC	0.799 ± 0.000	**0.693 ± 0.000**

Two datasets (German credit and Adult income) come from the UCI ML repository (Frank, Asuncion, et al., 2010), whereas the Fetal health dataset is a publicly available dataset that can be found on Kaggle. We summarize the properties of the datasets that we use in Table 2.

In the *Adult income* dataset (Kohavi, 1996) the goal is to predict whether an income is greater than 50 K dollars. In the *German credit* dataset, the goal is to classify bank account holders into two classes—good or bad. In the *Fetal health* dataset, the goal is to predict whether a fetus is healthy or not based on the features extracted from cardiotocogram examination.

We compared MoËT with other supervised learning models which would require similar effort and tools to be verified: decision tree, support vector classifier (SVC) with linear kernel, ridge logistic regression and lasso logistic regression. The results are evaluated by F1 score and accuracy. The hyperparameters of compared models are tuned on validation set. The results evaluated on test set with 95% confidence intervals for *Fetal health*, *German credit*, and *Adult income* datasets are presented in Tables 3, 4, and 5, respectively. It can be observed that MoËT is the best performing model with exception of SVC being better on German credit data according to accuracy (but not F1 score). Therefore, it can be concluded that MoËT can also be successfully applied in the case of supervised learning problems.

7. Conclusion

We introduced MoËT, a technique based on MoE with decision trees as experts and formulated a learning algorithm to train MoËT models. To the best of our knowledge, this approach is the first to combine standard non-differentiable DT experts with MoE approach. Furthermore, we used MoËT in RL setting by mimicking DRL agents, in this way constructing RL policies that

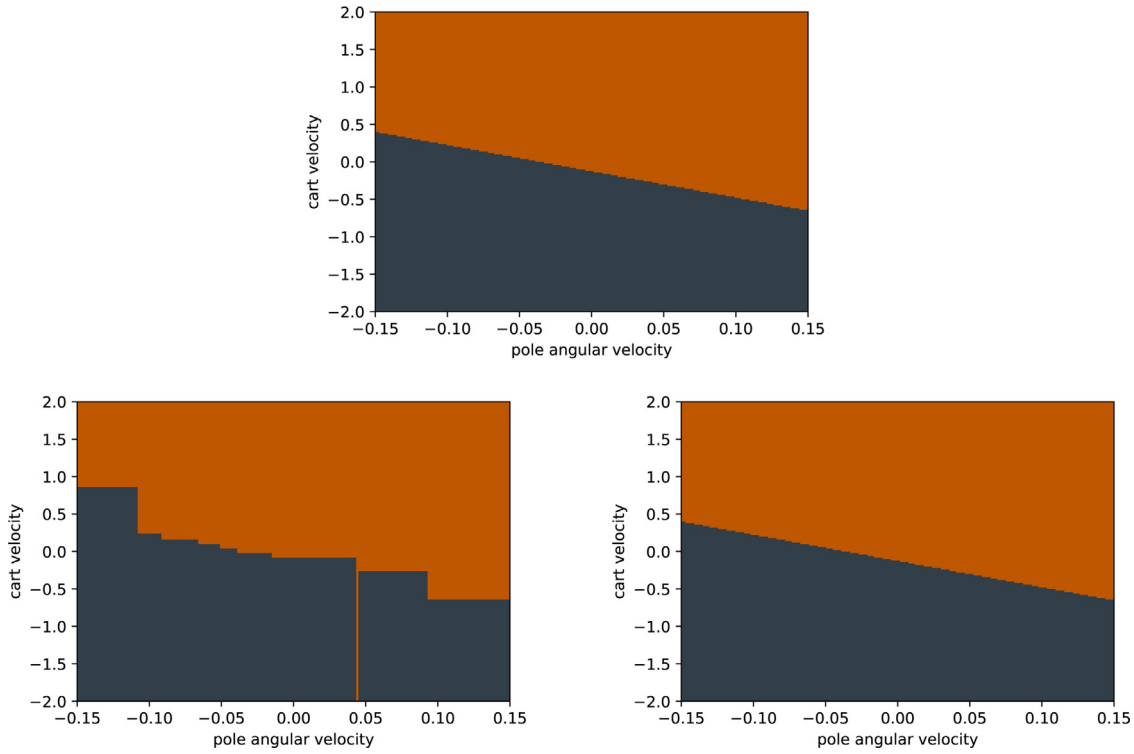

Fig. 6. Visualizing DRL (top), Viper (bottom left) and MoËT$_h$ (bottom right) policies on CartPole. *X*-axis represents pole angular velocity and *y*-axis cart velocity, which are the most discriminatory features (topmost nodes in the Viper decision tree policy). Other features, cart position and pole angle, are set to 0 (center position with pole upright). Gray color represents points where agent takes action *left*, and orange points when agent takes action *right*. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
Prediction performance of classifiers—*Adult income* dataset.

Model/metrics	F1 score	Accuracy
Decision tree	0.661 ± 0.003	0.852 ± 0.001
Lasso logistic regression	0.536 ± 0.000	0.820 ± 0.000
MoËT$_h$	**0.676 ± 0.000**	0.854 ± 0.000
MoËT	**0.674 ± 0.004**	**0.860 ± 0.001**
Ridge logistic regression	0.529 ± 0.000	0.819 ± 0.000
SVC	0.406 ± 0.000	0.805 ± 0.000

can be verified and are more interpretable than the DRL agents themselves. We showed a procedure to translate MoËT policies into SMT logic providing rich means for verification, and showed that MoËT models perform better than the previous state-of-the-art approach Viper and that they are also useful in the supervised regime.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by NSF, United States grant CCF-1718903 to SK. We thank Professor Risto Miikkulainen and Professor Peter Stone for constructive discussions and feedback on this work.

Appendix A. Viper algorithm

Viper algorithm is shown in Algorithm 2.

Appendix B. Environments

In this section we provide a brief description of environments we used in our experiments. We used five environments from OpenAI Gym: CartPole, Acrobot, Mountaincar, Lunarlander, Pong and Pendulum.

B.1. CartPole

This environment consists of a cart and a rigid pole hinged to the cart, based on the system presented by Barto, Sutton, and Anderson (1983). At the beginning pole is upright, and the goal is to prevent it from falling over. Cart is allowed to move horizontally within predefined bounds, and controller chooses to apply either *left* or *right* force to the cart. State is defined with four variables: x (cart position), $\dot{x}$ (cart velocity), θ (pole angle), and $\dot{\theta}$ (pole angular velocity). Game is terminated when the absolute value of pole angle exceeds 12°, cart position is more than 2.4 units away from the center, or after 200 successful steps; whichever comes first. In each step reward of $+1$ is given, and the game is considered solved when the average reward is over 195 in over 100 consecutive trials.

B.2. Acrobot

This environment is analogous to a gymnast swinging on a horizontal bar, and consists of a two links and two joins, where the joint between the links is actuated. The environment is based on the system presented by Sutton (1996). Initially both links are pointing downwards, and the goal is to swing the end-point (feet) above the bar for at least the length of one link. The state consists of six variables, four variables consisting of sin and cos values of the joint angles, and two variables for angular velocities of the joints. The action is either applying *negative*, *neutral*, or *positive*

Algorithm 2 Viper training (Bastani et al., 2018).

procedure VIPER (MDP e, TEACHER π_t, Q-FUNCTION Q^{π_t}, ITERATIONS N)
  Initialize dataset and student: $D \leftarrow \emptyset$, $\pi_{s_0} \leftarrow \pi_t$
  for $i \leftarrow 1$ to N **do**
    Sample trajectories and aggregate: $D \leftarrow D \cup \{(s, \pi_t(s)) \sim d^{\pi_{s_{i-1}}}(e)\}$
    Sample dataset using Q values: $D_s \leftarrow \{(s, a) \in I \sim D\}$
    Train decision tree: $\pi_{s_i} \leftarrow fit\_tree(D_s)$
  return Best policy $\pi_s \in \{\pi_{s_1}, ..., \pi_{s_N}\}$.

torque on the joint. At each time step reward of -1 is received, and episode is terminated upon successful reaching the height, or after 200 steps, whichever comes first. Acrobot is an unsolved environment in that there is no reward limit under which is considered solved, but the goal is to achieve high reward.

B.3. Mountaincar

This environment consists of a car positioned between two hills, with a goal of reaching the hill in front of the car. The environment is based on the system presented by Moore (1990). Car can move in a one-dimensional track, but does not have enough power to reach the hill in one go, thus it needs to build momentum going back and forth to finally reach the hill. Controller can choose *left*, *right* or *neutral* action to apply left, right or no force to the car. State is defined by two variables, describing car position and car velocity. In each step reward of -1 is received, and episode is terminated upon reaching the hill, or after 200 steps, whichever comes first. The game is considered solved if average reward over 100 consecutive trials is no less than -110.

B.4. Lunarlander

This environment consists of a space ship and a landing pad, to which the ship should land. Controller can choose when to turn on the left engine, right engine or the main engine, thus controlling the movement of the ship. State is defined by: x and y coordinates of the lander, v_x and v_y velocities in the x and y direction, θ angle of the lander, α angular velocity, and two boolean values indicating if left or right leg is touching the ground. Episode finishes when lander crashes or comes to rest, after which it received appropriate reward. Firing main engine is -0.3 points, and each leg contact is 10 points. The game is considered solved if achieved reward is at least 200 points.

B.5. Pong

This is a classical Atari game of table tennis with two players. Minimum possible score is -21 and maximum is 21.

B.6. Pendulum

The environment consists of a pendulum, and the goal is to swing it up so it stays upright. State is defined by: θ—angle of the pendulum, and ω—angular velocity of the pendulum. Note that the OpenAI gym environment instead of the state feature θ contains two features: x (which is equal to $cos(\theta)$) and y (which is equal to $sin(\theta)$). Action available is applying torque to the pendulum. In OpenAI gym action can take any value in range $[-2, 2]$. We discretize action space into 3 possible actions corresponding to torque of -2, 0, or 2. In each step reward obtained is equal to $-(\theta^2 + 0.1cdot\omega^2 + 0.001 \cdot torque^2)$. Thus, the maximum reward that can be obtained in a step is 0, which occurs when pendulum is upright, with zero velocity, and 0 torque is applied to the pendulum. Episode is of length 200.

Appendix C. Model training parameters

C.1. DRL agent training

In this section we present the architectures and hyperparameters used to train DRL agents for different environments.

For CartPole, we use policy gradient model as used in Viper. While we use the same model, we had to retrain it from scratch as the trained Viper agent was not available. We use 1 hidden layer with 8 neurons. We set discount factor to 0.99, number of epochs to 1000 and batch size to 50.

For Pong, we use a DQN network (Mnih et al., 2015) model that is already trained (the same as used in Viper). This model originates from the OpenAI baselines (OpenAI, 0000).

For Acrobot, Mountaincar and Lunarlander, we implement our own version of dueling DQN network following (Wang et al., 2015). We use 3 hidden layers with 15 neurons in each layer for Mountaincar, and 50 neurons in each layer for Acrobot and Lunarlander. We set the learning rate to 0.001, batch size to 30 in Mountaincar, 50 in Acrobot and Lunarlander, step size to 10,000 and number of epochs to 80,000 in Mountaincar, 50,000 in Acrobot and Lunarlander. We checkpoint a model every 5000 steps and pick the best performing one in terms of achieved reward.

C.2. Viper and MoËT training

We used 40 iterations of DAGGER, and 200,000 as a maximum number of samples for training student policies. During evaluation, cumulative reward is averaged across 100 runs in a given environment (250 in a case of CartPole).

We trained Viper for varying value of the tree maximum depth. The values used are: [1, 15] in CartPole, [1, 20] in Acrobot, [1, 20] in Mountaincar, [1, 30] In Lunarlander, and [1, 35] in Pong.

We trained MoËT models for varying number of experts and their maximum depths. The number of experts used are: [2, 8] in CartPole, $[2, 8] \cup [15, 16]$ in Acrobot, $[2, 8] \cup \{12, 16\}$ in Mountaincar, [2, 8] in Lunarlander, and $\{2, 4, 8, 16, 32\}$ in Pong. The maximum depths of experts are: [0, 7] in CartPole, [0, 15] in Acrobot, [0, 11] in Mountaincar, [0, 20] in Lunarlander, and [0, 29] in Pong. We used following learning rates for training MoËT models: $\{1, 0.3, 0.1, 0.01, 0.001, 0.0001, 0.00001\}$, while for the learning rate decay we used 1 (no decay) and 0.97 (learning rate is multiplied by this value after each epoch). As for the maximum number of epochs for MoËT training procedure we used values: $\{50, 100, 500\}$.

C.3. Compute

To run our experiments we used a cluster with nodes of the following configuration: Xeon CPU E5-2650 v3 (Haswell): 10 cores per socket (20 cores/node), 2.30 GHz, 128 GB DDR4-2133. We used up to 10 such nodes when scheduling our experiments.

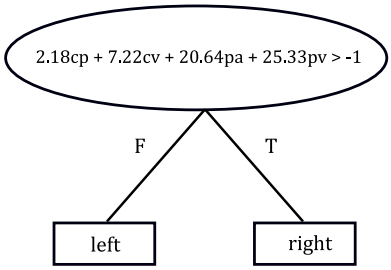

Fig. D.7. Example CartPole MoËT$_h$ policy.

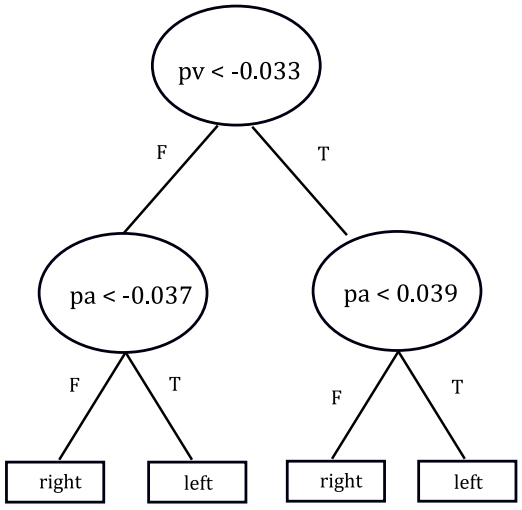

Fig. D.8. Example CartPole Viper policy.

Table E.6
CartPole: Details of models on the Global Pareto front shown in Fig. 3.

Model	Configuration	Reward	Fidelity
MoËT	E2-D0	200.00	0.998

Appendix D. SMT translation example

The CartPole MoËT$_h$ policy presented in Fig. 6 is shown in Fig. D.7. SMT formula that would encode the policy part (mapping input to a model decision) of CartPole verification formula would look as follows: `If(2.18cp + 7.22cv + 20.64pa + 25.33pv > -1, 1, 0)`. This MoËT$_h$ policy consists of the gating expressed by the inequality and two trivial expert decision trees of depth 0. Therefore, the second and third part of the `If` formula are trivial. In case that decision trees were nontrivial, those parts of the formula would be expanded with nested if expressions.

A simple depth 2 Viper policy for CartPole is shown in Fig. D.8. SMT formula that would encode the policy part of this formula would look like following: `If(pv < -0.033, If(pa < 0.039, 0, 1), If(pa < -0.037, 0, 1))`

The full formula for CartPole environment verification contains additional details, it is the conjunction of the formula encoding the policy, the safety requirements and the environment dynamics, as illustrated by the formula in Section 5.

Appendix E. Evaluation results

Tables E.6–E.11 show data about models on the global Pareto front presented in Fig. 3 of Section 6.

Table E.7
Acrobot: Details of models on the Global Pareto front shown in Fig. 3.

Model	Configuration	Reward	Fidelity
MoËT	E16-D11	−72.12	0.936
MoËT	E15-D11	−71.95	0.936
MoËT	E15-D11	−71.81	0.921
MoËT	E16-D9	−71.67	0.921
MoËT	E16-D0	−69.83	0.916
MoËT	E16-D0	−68.68	0.907

Table E.8
Mountaincar: Details of models on the Global Pareto front shown in Fig. 3.

Model	Configuration	Reward	Fidelity
MoËT$_h$	E6-D9	−107.00	0.984
MoËT	E6-D7	−106.83	0.984
MoËT	E16-D7	−105.90	0.983
MoËT	E7-D8	−104.28	0.982
MoËT	E3-D7	−103.86	0.979
MoËT	E3-D10	−103.82	0.977
MoËT$_h$	E3-D6	−103.77	0.977
MoËT	E7-D5	−103.75	0.974
MoËT	E3-D7	−103.22	0.973
Viper	D12	−102.83	0.973
MoËT	E2-D8	−102.45	0.972
Viper	D11	−102.05	0.972
MoËT$_h$	E4-D4	−101.40	0.971
MoËT	E5-D5	−101.09	0.966
MoËT$_h$	E8-D5	−100.97	0.962
MoËT$_h$	E4-D5	−100.96	0.961
MoËT$_h$	E2-D8	−100.95	0.961
MoËT$_h$	E4-D5	−98.85	0.960
MoËT$_h$	E4-D5	−98.70	0.950
MoËT	E4-D4	−97.84	0.943
Viper	D5	−97.46	0.938
MoËT	E7-D2	−97.39	0.922
MoËT	E4-D2	−96.96	0.914
MoËT$_h$	E6-D1	−96.78	0.912

Table E.9
Lunarlander: Details of models on the Global Pareto front shown in Fig. 3.

Model	Configuration	Reward	Fidelity
MoËT	E8-D17	204.13	0.792
MoËT	E7-D17	210.79	0.767
MoËT	E8-D17	217.33	0.765
MoËT	E8-D17	225.24	0.755
MoËT$_h$	E8-D17	229.20	0.747
MoËT	E6-D17	230.67	0.743
MoËT$_h$	E7-D0	239.96	0.666
MoËT$_h$	E7-D0	241.25	0.635
MoËT	E6-D3	253.64	0.628
MoËT$_h$	E7-D0	261.86	0.547

Table E.10
Pong: Details of models on the Global Pareto front shown in Fig. 3.

Model	Configuration	Reward	Fidelity
MoËT	E16-D21	21.00	0.896

Table E.11
Pendulum: Details of models on the Global Pareto front shown in Fig. 3.

Model	Configuration	Reward	Fidelity
MoËT	E8-D16	−170.00	0.988
MoËT$_h$	E7-D17	−141.17	0.988
MoËT	E4-D15	−134.06	0.988
MoËT	E6-D13	−127.25	0.985
MoËT$_h$	E2-D12	−120.31	0.979

Tables E.12–E.17 show data about the models on the global Pareto after re-evaluation is performed. This corresponds to data presented in Fig. 4 of Section 6.

Table E.12
CartPole: Details of models on the Global Pareto front shown in Fig. 4.

Model	Configuration	Reward	Fidelity
MoËT	E2-D0	200.00	0.998

Table E.13
Acrobot: Details of models on the Global Pareto front shown in Fig. 4.

Model	Configuration	Reward	Fidelity
MoËT	E15-D11	−76.31	0.923
MoËT	E15-D11	−75.98	0.920
MoËT	E16-D11	−75.81	0.934
MoËT	E16-D9	−72.12	0.911
MoËT	E16-D0	−70.67	0.909
MoËT	E16-D0	−70.66	0.907

Table E.14
Mountaincar: Details of models on the Global Pareto front shown in Fig. 4.

Model	Configuration	Reward	Fidelity
MoËT	E3-D7	−108.52	0.970
MoËT	E7-D8	−107.44	0.981
MoËT	E16-D7	−107.00	0.981
MoËT	E3-D7	−106.46	0.976
MoËT	E3-D10	−106.44	0.976
MoËT	E6-D7	−106.14	0.983
MoËT_{h}	E3-D6	−106.09	0.973
MoËT_{h}	E6-D9	−106.02	0.979
Viper	D11	−105.82	0.968
MoËT	E2-D8	−105.72	0.970
Viper	D12	−105.43	0.969
MoËT	E7-D5	−103.72	0.972
MoËT_{h}	E8-D5	−102.92	0.958
MoËT_{h}	E2-D8	−102.81	0.960
MoËT	E5-D5	−101.83	0.961
MoËT_{h}	E4-D5	−101.75	0.960
MoËT_{h}	E4-D4	−101.17	0.968
MoËT_{h}	E6-D1	−99.82	0.906
MoËT	E4-D2	−99.47	0.910
MoËT	E4-D4	−99.37	0.936
MoËT_{h}	E4-D5	−99.28	0.956
MoËT	E7-D2	−99.14	0.914
Viper	D5	−98.20	0.937
MoËT_{h}	E4-D5	−97.88	0.950

Table E.15
Lunarlander: Details of models on the Global Pareto front shown in Fig. 4.

Model	Configuration	Reward	Fidelity
MoËT	E8-D17	178.93	0.762
MoËT	E6-D17	180.40	0.751
MoËT_{h}	E8-D17	180.93	0.754
MoËT	E8-D17	185.42	0.765
MoËT	E7-D17	201.25	0.742
MoËT	E8-D17	202.76	0.756
MoËT_{h}	E7-D0	232.45	0.660
MoËT_{h}	E7-D0	240.48	0.660
MoËT_{h}	E7-D0	247.97	0.537
MoËT	E6-D3	256.90	0.588

Table E.16
Pong: Details of models on the Global Pareto front shown in Fig. 4.

Model	Configuration	Reward	Fidelity
MoËT	E16-D21	21.00	0.898

Table E.17
Pendulum: Details of models on the Global Pareto front shown in Fig. 4.

Model	Configuration	Reward	Fidelity
MoËT_{h}	E2-D12	−177.01	0.976
MoËT_{h}	E7-D17	−169.55	0.988
MoËT	E4-D15	−166.47	0.986
MoËT	E6-D13	−146.85	0.982
MoËT	E8-D16	−130.11	0.987

References

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML*.

Amir, G., Schapira, M., & Katz, G. (2021). Towards scalable verification of deep reinforcement learning. In *2021 formal methods in computer aided design (FMCAD)* (pp. 193–203). IEEE.

Ayala, A., Cruz, F., Fernandes, B., & Dazeley, R. (2021). Explainable deep reinforcement learning using introspection in a non-episodic task. arXiv preprint arXiv:2108.08911.

Bacci, E., & Parker, D. (2022). Verified probabilistic policies for deep reinforcement learning. arXiv preprint arXiv:2201.03698.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, (5), 834–846.

Bastani, O., Pu, Y., & Solar-Lezama, A. (2018). Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems* (pp. 2499–2509).

Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), (2009). *Frontiers in artificial intelligence and applications*: *vol. 185*, *Handbook of Satisfiability*. IOS Press.

Breiman, L., & Shang, N. (1996). *Born again trees*: *Technical Report 1*, (p. 2). Berkeley, Berkeley, CA: University of California.

Buciluǎ, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 535–541). ACM.

Cheng, J.-Z., Ni, D., Chou, Y.-H., Qin, J., Tiu, C.-M., Chang, Y.-C., et al. (2016). Computer-aided diagnosis with deep learning architecture: applications to breast lesions in us images and pulmonary nodules in ct scans. *Scientific Reports*, *6*(1), 1–13.

Cicero, M., Bilbily, A., Colak, E., Dowdell, T., Gray, B., Perampaladas, K., et al. (2017). Training and validating a deep convolutional neural network for computer-aided detection and classification of abnormalities on frontal chest radiographs. *Investigative Radiology*, *52*(5), 281–287.

De Moura, L., & Bjørner, N. (2008). Z3: An efficient SMT solver. In *International conference on tools and algorithms for the construction and analysis of systems* (pp. 337–340). Springer.

Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608.

Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., et al. (2019). A guide to deep learning in healthcare. *Nature Medicine*, *25*(1), 24–29.

Frank, A., Asuncion, A., et al. (2010). Uci machine learning repository. URL http://archive.ics.uci.edu/ml 15 (2011) 22.

Furlanello, T., Lipton, Z. C., Tschannen, M., Itti, L., & Anandkumar, A. (2018). Born again neural networks. arXiv preprint arXiv:1805.04770.

Gao, Z., Xu, K., Ding, B., & Wang, H. (2021). Knowru: Knowledge reuse via knowledge distillation in multi-agent reinforcement learning. *Entropy*, *23*(8), 1043.

Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge distillation: A survey. *International Journal of Computer Vision*, *129*(6), 1789–1819.

Guidotti, R., Monreale, A., Ruggieri, S., Pedreschi, D., Turini, F., & Giannotti, F. (2018). Local rule-based explanations of black box decision systems. arXiv preprint arXiv:1805.10820.

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., & Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM Computing Surveys*, *51*(5), 93.

Hester, T., & Stone, P. (2013). Texplore: real-time sample-efficient reinforcement learning for robots. *Machine Learning*, *90*(3), 385–429.

Heuillet, A., Couthouis, F., & Díaz-Rodríguez, N. (2021). Explainability in deep reinforcement learning. *Knowledge-Based Systems*, *214*, Article 106685.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

Irsoy, O., Yıldız, O. T., & Alpaydın, E. (2012). Soft decision trees. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)* (pp. 1819–1822). IEEE.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., Hinton, G. E., et al. (1991). Adaptive mixtures of local experts.. *Neural Computation*, *3*(1), 79–87.

Jordan, M. I., & Xu, L. (1995). Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, *8*(9), 1409–1431.

Kazak, Y., Barrett, C., Katz, G., & Schapira, M. (2019). Verifying deep-rl-driven systems. In *Proceedings of the 2019 workshop on network meets AI & ML* (pp. 83–89).

Kohavi, R. (1996). Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, *96* (pp. 202–207).

Kooi, T., Litjens, G., Van Ginneken, B., Gubern-Mérida, A., Sánchez, C. I., Mann, R., et al. (2017). Large scale deep learning for computer aided detection of mammographic lesions. *Medical Image Analysis*, *35*, 303–312.

Kotsiantis, S. B. (2013). Decision trees: a recent overview. *Artificial Intelligence Review*, *39*(4), 261–283.
Koul, A., Fern, A., & Greydanus, S. (2019). Learning finite state representations of recurrent policy networks. In *ICLR*.
Li, X., Serlin, Z., Yang, G., & Belta, C. (2019). A formal methods approach to interpretable reinforcement learning for robotic planning. *Science Robotics*, *4*(37).
Lipton, Z. C. (2016). The mythos of model interpretability. arXiv preprint arXiv:1606.03490.
Miotto, R., Wang, F., Wang, S., Jiang, X., & Dudley, J. T. (2018). Deep learning for healthcare: review, opportunities and challenges. *Briefings in Bioinformatics*, *19*(6), 1236–1246.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.
Moore, A. W. (1990). Efficient memory-based learning for robot control.
Niu, C., Wu, F., Tang, S., Ma, S., & Chen, G. (2020). Toward verifiable and privacy preserving machine learning prediction. *IEEE Transactions on Dependable and Secure Computing*.
Niuniu, X., & Yuxun, L. (2010). Notice of retraction: Review of decision trees. In *2010 3rd international conference on computer science and information technology, Vol. 5* (pp. 105–109). IEEE.
OpenAI Baselines, https://github.com/openai/baselines.
Puiutta, E., & Veith, E. (2020). Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction* (pp. 77–95). Springer.
Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should I trust you?: Explaining the predictions of any classifier. In *KDD*.
Roscher, R., Bohn, B., Duarte, M. F., & Garcke, J. (2020). Explainable machine learning for scientific insights and discoveries. *IEEE Access*, *8*, 42200–42216.
Ross, S., Gordon, G., & Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 627–635).
Rusu, A. A., Colmenarejo, S. G., Gülçehre, Ç., Desjardins, G., Kirkpatrick, J., Pascanu, R., et al. (2016). Policy distillation. In *4th international conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference track proceedings*.
Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*.
Shi, S., Li, J., Li, G., Pan, P., & Liu, K. (2021). Xpm: An explainable deep reinforcement learning framework for portfolio management. In *Proceedings of the 30th ACM international conference on information & knowledge management* (pp. 1661–1670).
Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484.
Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems* (pp. 1038–1044).
Törnblom, J., & Nadjm-Tehrani, S. (2018). Formal verification of random forests in safety-critical applications. In *International workshop on formal techniques for safety-critical systems* (pp. 55–71). Springer.
Törnblom, J., & Nadjm-Tehrani, S. (2020). Formal verification of input–output mappings of tree ensembles. *Science of Computer Programming*, *194*, Article 102450.
Tsantekidis, A., Passalis, N., & Tefas, A. (2021). Diversity-driven knowledge distillation for financial trading using deep reinforcement learning. *Neural Networks*, *140*, 193–202.
Van Wesel, P., & Goodloe, A. E. (2017). *Challenges in the verification of reinforcement learning algorithms*: *Tech. rep.*.
Verma, A., Le, H. M., Yue, Y., & Chaudhuri, S. (2019). Imitation-projected programmatic reinforcement learning. arXiv preprint arXiv:1907.05431.
Wang, J., & Pandit, S. (2019). Towards high-level, verifiable autonomous behaviors with temporal specifications. In *2019 IEEE national aerospace and electronics conference (NAECON)* (pp. 92–99). IEEE.
Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.
Wang, Z., Wei, Y., & Wu, F. (2021). Knowledge distillation based cooperative reinforcement learning for connectivity preservation in uav networks. In *2021 international conference on UK-China emerging technologies (UCET)* (pp. 171–176). IEEE.
Wells, L., & Bednarz, T. (2021). Explainable ai and reinforcement learning—a systematic review of current approaches and trends. *Frontiers in Artificial Intelligence*, *4*, 48.
Yuksel, S. E., Wilson, J. N., & Gader, P. D. (2012). Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(8), 1177–1193.
Zhang, A., Lipton, Z. C., Pineda, L., Azizzadenesheli, K., Anandkumar, A., Itti, L., et al. (2019). Learning causal state representations of partially observable environments. CoRR.
Zhao, W., Gao, Y., Memon, S. A., Raj, B., & Singh, R. (2019). Hierarchical routing mixture of experts. arXiv preprint arXiv:1903.07756.
Zhu, H., Xiong, Z., Magill, S., & Jagannathan, S. (2019). An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation* (pp. 686–701).