Non-divergent Imitation for Verification of Complex Learned Controllers

Vahdat Abdelzad*, Jaeyoung Lee*, Sean Sedwards*, Soheil Soltani* and Krzysztof Czarnecki
University of Waterloo, Canada
{vahdat.abdelzad, jaeyoung.lee, sean.sedwards, soheil.soltani, krzysztof.czarnecki}@uwaterloo.ca

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533410

***Abstract*—We consider the problem of verifying complex learned controllers using distillation. In contrast to previous work, we require that the distilled model maintains behavioural fidelity with an oracle, defining the notion of *non-divergent path length* (NPL) as a metric. We demonstrate that current distillation approaches with proven accuracy bounds do not have high expected NPL and can be out-performed by naive behavioural cloning. We thus propose a distillation algorithm that typically gives greater expected NPL, improved sample efficiency, and more compact models. We prove properties of NPL maximization and demonstrate the performance of our algorithm on deep Q-network controllers for three standard learning environments that have been used in this context: Pong, CartPole and MountainCar.**

***Index Terms*—imitation learning, behavioural fidelity, verification, distillation, reinforcement learning, DQN, decision tree**

I. Introduction

Machine learning can solve complex sequential decision-making tasks, but learned controllers are often opaque and difficult to formally verify. If the system is sufficiently simple and it is possible to constrain its architecture and training, it may be feasible to directly verify a learned controller [1–3]. In this work, however, we consider the more common case that the controller already exists and is not amenable to direct formal analysis. We thus pursue the idea of distilling a complex deterministic controller—an *oracle*—into a data structure that is more tractable to verification. In particular, we consider the process of distilling such an oracle into a decision tree (DT), as recently proposed in [4], but treating the oracle as a black box. A symbolic rollout of the execution of a DT can be represented as a set of simple constraints, which can then be verified w.r.t. linear temporal logic (LTL) using bounded model checking (BMC) [5].

This approach raises the issue of fidelity, i.e., the ability of the distilled model to achieve similar expected rewards (performance) and to make similar decisions (accuracy) to the original controller. The "Viper" distillation algorithm of [4] inherits the proven fidelity bounds of the "Dagger" imitation learning algorithm [6], however these bounds are only asymptotic w.r.t. execution path length. By its nature, BMC checks properties of finite length, which are typically short for reasons of computability. We thus find that Dagger's and Viper's fidelity are not adequate for verification.

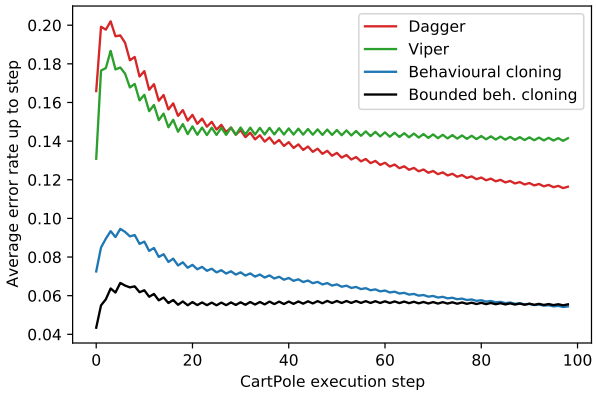


Fig. 1: Average error rate $(1 - \text{accuracy})$ w.r.t. the oracle up to given execution step, for competing distillation approaches applied to the CartPole problem. Averages over 10 distillations. 10000 rollouts used to estimate errors.

Using a deep Q-network (DQN) controller for the CartPole problem [7] as an example, Fig. 1 illustrates the motivation of our work by plotting the error rate $(1 - \text{accuracy})$ up to a given execution step for DTs produced by Dagger, Viper, naive behavioural cloning and behavioural cloning with bounded executions, all using the same total amount of training data. We see that Dagger and Viper have much higher error rates than behavioural cloning, with significant peaks before 20 steps, noting that a distilled model of CartPole is verified for only 10 steps in [4]. Bounded behavioural cloning, in this case where the DT is trained on states from executions limited to 40 steps, performs better than naive behavioural cloning. The intuition is that accuracy is improved by focusing on the states that matter, however accuracy is not a sufficient metric for verification. LTL properties typically apply to sequences of states generated by the execution of a system from a distribution of initial states, while accuracy is a metric of fidelity between states and actions, irrespective of the sequence. Hence, it is possible to have high accuracy on a sequence of states that is not in a valid order. In Section III, we therefore define the notion of *non-divergent path length* (NPL), which can be seen as the number of steps in an execution sequence before taking a different

*Contributed equally.

The authors gratefully acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) DND Supplement and the Japanese Science and Technology agency (JST) ERATO project JPMJER1603: HASUO Metamathematics for Systems Design.

action to a reference sequence. After such a divergence, the sequence of states cannot be guaranteed to follow the reference, even if the subsequent actions match.

We thus propose statistics over NPL as suitable metrics to judge the behavioural fidelity of a distilled model w.r.t. an oracle. We then devise an imitation learning algorithm that uses NPL to increase the behavioural fidelity of distillation. Our approach is therefore in contrast to standard imitation learning, which typically expects, allows, and mitigates divergence from the oracle. Our algorithm tries to avoid divergence in the first place.

The paper proceeds as follows. We first briefly summarize related work in Section II. In Section III, we define notation and formulate our metric (NPL) and the problem we address. In Section IV, we give theoretical insights about NPL maximization and prove related properties. We describe our non-divergent imitation (NDI) algorithm in Section V, and give further theoretical insights about how it works. In Section VI, we give the results of distillation experiments performed on DQN controllers for standard learning environments. Finally, we summarize our contribution in Section VII.

II. Related Work

Imitation Learning: Also known as learning from demonstration, imitation learning attempts to learn a model that mimics the behaviour of an oracle. The learned models can be complex parameterized models, such as deep neural networks [6, 8–10], interpretable models, such as DTs [4, 11, 12], or synthetic programs [12]. In this work, we are interested in approaches where the learned models have a data structure that is tractable to verification. The classes of imitation learning relevant to our work are behavioural cloning [8] and direct policy optimization [4, 6, 8], where the data distributions come from rollouts by an oracle and learned models, respectively. Dagger [6] is the standard direct policy optimizer. Viper [4] is a grey-box algorithm that extends Dagger by introducing selective data sampling based on Q-values. See [11] for a further extension using a mixture of expert trees.

Dagger-style imitation was applied as a projection operation in a meta-algorithm Propel [12], which alternates re-training the oracle and projecting it into a DT or program space. Viper was used to construct a shield for a monitoring system [13]. While the purpose of a shield is to monitor and protect the deployed system online, the philosophy of our work is to verify the controller, before its deployment.

Verification of Deep Q-networks: Much current work focuses on verifying controllers in the form of deep Q-networks (DQN). This can be divided into direct verification [1–3, 14] and verification of distilled models [4, 11–13, 15, 16]. We adopt the latter perspective. The authors of [4, 11–13, 15] have demonstrated the use of the Z3 SMT solver [17] to verify their distilled DTs. The authors of Viper [4] show examples of stability- and robustness-checking of distilled DTs, finding that the latter is faster than directly checking the robustness of the DQN oracle with Reluplex [1]. However, despite titles that sometimes imply the contrary, these works do not actually verify the DQN and do not consider the behavioural fidelity of the DT w.r.t. the DQN. Without such consideration, the DQN cannot be confidently deployed, because the verification of the distilled DT will not imply the verification of the DQN.

III. Problem Formulation

We consider a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, d, \mathcal{P})$ for a state space $\mathcal{S}$, a finite action space $\mathcal{A}$, an initial state distribution $d \in \mathrm{Dist}(\mathcal{S})$, and a state transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathrm{Dist}(\mathcal{S})$ describing the distribution of the next state given the current state and chosen action. $\mathrm{Dist}(\mathcal{S})$ denotes the set of all distributions over $\mathcal{S}$. For clarity, our theory below assumes that $\mathcal{S}$ is countable, but can easily be extended to MDPs where $\mathcal{S}$ is Euclidean space.

We focus on episodic tasks where an underlying MDP generates a *finite* path $\tau := s_1 a_1 s_2 a_2 \cdots s_{|\tau|} a_{|\tau|}$ s.t. $|\tau| = T$, $s_1 \sim d(\cdot)$, and $s_{n+1} \sim \mathcal{P}(s_n, a_n)(\cdot)$ for all $n = 1, \cdots, T-1$; $|\tau|$ denoted the length of τ; $T \in \mathbb{N}$ is the *terminal index* that is bounded with probability 1 and determined by either reaching a terminal state or timeout. We say τ is *complete* if $|\tau| = T$. Given a path τ, we adopt the notation $\tau_{1:t} := s_1 a_1 \cdots s_t a_t$ and $\tau_{t:t} := s_t a_t$ for $t \le |\tau|$, and $\tau_{1:t} := \tau$ for $t > |\tau|$. Similarly, $s_{1:t} := s_1 \cdots s_t$ and $a_{1:t} := a_1 \cdots a_t$ for $t \le |\tau|$, etc.

A *policy* π in our work refers to a mapping from $\mathcal{S}$ to $\mathcal{A}$. τ^π denotes a complete path generated by π. Given a policy π and a path τ, the following notation will be used for $t \in \mathbb{N}$:

$$\pi(s_{1:t}) = a_{1:t} \iff \pi(s_n) = a_n \ \ \forall n \in \{1, 2, \cdots, \min(t, |\tau|)\}.$$

Note: $\pi(s_{1:t}) = a_{1:t}$ is reduced to $\pi(s_{1:|\tau|}) = a_{1:|\tau|}$ if $t > |\tau|$.

The *non-divergent path length (NPL)* $\mathit{l} \equiv \mathit{l}(\pi|\tau)$ of a policy π given a path τ is defined as:

$$\begin{aligned} \mathit{l}(\pi|\tau) &:= \max\left\{t \in \mathbb{N}_0 \mid t = 0 \text{ or } \pi(s_{1:t}) = a_{1:t}\right\} \\ &= \textstyle\sum_{t=1}^{|\tau|} \mathbf{1}\big[\pi(s_{1:t}) = a_{1:t}\big], \end{aligned} \tag{1}$$

where $\mathbf{1}[\cdot]$ is the indicator function. Intuitively, the NPL $\mathit{l}(\pi|\tau)$ is a measure of how long the given policy π can generate the same sequence of actions from the initial state as the ones taken in the path τ. If $\mathit{l} < |\tau|$, then the NPL l indicates

$$\pi(s_{1:\mathit{l}}) = a_{1:\mathit{l}} \text{ but } \pi(s_{\mathit{l}+1}) \neq a_{\mathit{l}+1}.$$

Hereafter, we use l for an NPL whenever there is no confusion and unless specified, τ denotes a complete path τ^{π^*} generated by an *oracle* π^*, with a slight abuse of notation.

Given an oracle π^* to be verified and a class of verifiable policies Π, our objective is to find a policy $\hat{\pi}$ such that

$$\hat{\pi} \in \arg\max_{\pi \in \Pi} \mathbb{E}\big[\mathit{l}(\pi|\tau)\big] \tag{2}$$

where the expectation $\mathbb{E}[\cdot]$ is taken w.r.t. all complete paths τ generated by π^*. Note that the higher $\mathbb{E}\big[\mathit{l}(\pi|\tau)\big]$, the more accurate imitation is expected *on* $\tau_{1:\mathit{l}}$ *in the sequential manner*, which is desirable for verifying the oracle π^*. On the other hand, the 0-1 loss in standard imitation learning (e.g., behavioural cloning and Dagger) does not have this property. Thus, reduction of the 0-1 loss results in better accuracy on

the entire space, but not necessarily in a sequential manner on $\tau_{1:\ell}$. Also note that the maximum NPL $|\tau|$ along a path τ is achievable at π^* that generates τ (see the maximality in Theorem 1 below). However, in practice, the oracle π^* is not in the constrained policy class Π hence $\hat{\pi} \neq \pi^*$ and the maximal achievable NPL $\ell(\pi|\tau)$ is less than $|\tau|$.

The framework (2) and our algorithm can be applied to any type of oracle π^* and any verifiable policy class Π. In our experiments, however, we consider a DQN policy π^* and a class of DT policies Π.

IV. Properties of NPL Maximization

In this section, we establish the properties of the NPL (1) and its maximization (2), which serves as theoretical basis of our proposed algorithm. The highlight of this section is that (2) is equivalent to the problem that our algorithm tries to solve. We start our discussion by showing the properties of the NPL and associated problems.

Theorem 1. *The NPL satisfies the following, for any path τ and any policies π and π'.*

1) $\ell(\pi|\tau^\pi) = |\tau^\pi|$ (maximality);

2) $\ell(\pi|\tau_{1:t_1}) \leq \ell(\pi|\tau_{1:t_2})$ for $t_1 \leq t_2$ (monotonicity);

3) $\mathbb{E}[\ell(\pi|\tau^{\pi'})] = \mathbb{E}[\ell(\pi'|\tau^\pi)]$ (equivalence).

Proof. The first and second parts are obvious by (1) and

$\pi(s_t) = a_t$ for any $t \in \{1, 2, \cdots, |\tau|\}$ if $\tau = \tau^\pi$,
$\sum_{t=1}^{t_1} \mathbf{1}\big[\pi(s_{1:t}) = a_{1:t}\big] \leq \sum_{t=1}^{t_2} \mathbf{1}\big[\pi(s_{1:t}) = a_{1:t}\big]$ if $t_1 \leq t_2$.

For the proof of equivalence, see Appendix A. □

By equivalence in Theorem 1, the problem (2) of maximizing the expected NPL can be transformed into a form similar to direct policy optimization:

$$\hat{\pi} \in \arg\max_{\pi \in \Pi} \mathbb{E}\big[\ell(\pi^*|\tau^\pi)\big] \tag{3}$$

which now depends on the complete path τ^π to be optimized. Maximizing (2) and (3) can be performed in a similar manner to standard imitation learning approaches (e.g., behavioural cloning, Dagger and Viper), where the paths are generated by the oracle π^* and the learned policy, respectively. However, (2) and (3) maximize the expected NPL while those imitation learning approaches minimize the pointwise 0-1 loss. It is worth noting that regardless of which policy (the oracle π^* or a learned policy) is employed to generate paths, the solution $\hat{\pi}$ remains to be the same by the equivalence between the problems (2) and (3). In our proposed algorithm, we adopt generating paths using the oracle π^* that is simple and produces reusable data over iterations, hence we consider the behavioural-cloning-style maximization (2) rather than (3).

The next equivalent problem, which is more relevant to our proposed algorithm, is defined w.r.t. the following *surrogate NPL* ℓ which we also call the *pathwise similarity* ℓ:

$$\ell(\pi|\tau) := \textstyle\sum_{t=1}^{|\tau|} \mathbf{1}\big[\pi(s_t) = a_t\big]. \tag{4}$$

While the condition "$\pi(s_{1:t}) = a_{1:t}$" for the NPL (1) is path-dependent, "$\pi(s_t) = a_t$" in (4) is not; $\mathbb{E}[\ell(\pi|\tau)]$ corresponds to the conventional 0-1 similarity. Here, the surrogate NPL ℓ is an upper-bound of or equivalent to the NPL ℓ, as shown in the next theorem.

Theorem 2. *For any path τ and any policy π,*

1) $\ell(\pi|\tau) \leq \ell(\pi|\tau) \leq |\tau|$ (boundedness);

2) $\ell(\pi|\tau_{1:t_1}) \leq \ell(\pi|\tau_{1:t_2})$ for $t_1 \leq t_2$ (monotonicity);

3) $\ell(\pi|\tau_{1:t}) = \ell(\pi|\tau_{1:t}) = t$ for $t \leq \ell(\pi|\tau)$ (equivalence).

Proof. The boundedness is obvious by (1), (4), and

$$\mathbf{1}\big[\pi(s_{1:t}) = a_{1:t}\big] \leq \mathbf{1}\big[\pi(s_t) = a_t\big] \leq 1.$$

For the monotonicity, note that for $t_1 \leq t_2$,

$$\textstyle\sum_{t=1}^{t_1} \mathbf{1}\big[\pi(s_t) = a_t\big] \leq \sum_{t=1}^{t_2} \mathbf{1}\big[\pi(s_t) = a_t\big].$$

Lastly, $\pi(s_n) = a_n$ for all $n \leq \ell(\pi|\tau)$. Thus, for $t \leq \ell(\pi|\tau)$,

$$\underbrace{\textstyle\sum_{n=1}^{t} \mathbf{1}[\pi(s_{1:n}) = a_{1:n}]}_{\ell(\pi|\tau_{1:t})} = \underbrace{\textstyle\sum_{n=1}^{t} \mathbf{1}[\pi(s_n) = a_n]}_{\ell(\pi|\tau_{1:t})} = \underbrace{\textstyle\sum_{n=1}^{t} 1}_{t}$$

which directly proves the equivalence, the last part. □

By the equivalence in Theorem 2, the primal problem (2) is equivalent to:

$$\hat{\pi} \in \arg\max_{\pi \in \Pi} \mathbb{E}\big[\ell(\pi \mid \tau_{1:\ell(\pi|\tau)})\big] \tag{5}$$

w.r.t. the pathwise similarity ℓ. Now, we can see that solving (2) is equivalent to finding the optimal path length, the NPL ℓ. Once ℓ is given for each path τ, the maximization (5) falls into the usual problem of minimizing the 0-1 loss on the data constructed from each roll-out $\tau_{1:\ell}$ but *only up to its optimal path length ℓ.*

The challenge in solving (5) here is that ℓ depends on the policy π we attempt to optimize. However, we can estimate it as discussed with our proposed algorithm in the next section. From Theorem 2 (see also monotonicity in Theorem 1), we observe that an estimate $\hat{l}$ of ℓ holds the following properties (note beforehand that $\ell \equiv \ell(\pi|\tau) = \ell(\pi|\tau_{1:\ell})$ is the objective to be maximized).

1) Consistency: $\begin{cases} \hat{l} \leq \ell \implies \ell(\pi|\tau_{1:\hat{l}}) \leq \ell \\ \ell \leq \hat{l} \implies \ell \leq \ell(\pi|\tau_{1:\hat{l}}) \end{cases}$

2) The closer $\hat{l}$ is to ℓ, the closer the associated NPL $\ell(\pi|\tau_{1:\hat{l}})$ and its surrogate $\ell(\pi|\tau_{1:\hat{l}})$ are to ℓ (due to monotonicity).

3) Equivalence: $\hat{l} \leq \ell \implies \ell(\pi|\tau_{1:\hat{l}}) = \ell(\pi|\tau_{1:\hat{l}}) = \hat{l}$.

V. Non-Divergent Imitation (NDI)

In this section, we present our Non-Divergent Imitation (NDI) algorithm (Algorithm 1), which generates $1 \leq i \leq i_{\max}$ verifiable policies π_i and returns the one with the greatest expected NPL. The core idea is to iteratively accumulate training data from the non-divergent prefixes of paths generated by a succession of improving models. In contrast to other imitation

Algorithm 1: Non-Divergent Imitation (NDI)

Input:
k : the number of paths per iteration
$i_{\max}$: the maximum number of iterations
$\pi_0 \equiv \pi^*$: the oracle policy
Output: A verifiable policy with high expected NPL

1 $\mathcal{T} \leftarrow \varnothing$ /* multiset of paths */
2 **for** $i = 1, \ldots, i_{\max}$ **do**
  /* initialize k new paths */
3 **repeat** k **times**
4 sample $s \sim d(\cdot)$ /* initial dist. */
5 $a \leftarrow \pi^*(s)$
6 $\mathcal{T} \leftarrow \mathcal{T} \cup \{sa\}$
7 $\mathcal{T}' \leftarrow \varnothing$ /* multiset of paths */
8 $D \leftarrow \varnothing$ /* multiset of state-acts. */
9 **foreach** $\tau = s_1 a_1 \cdots s_t a_t \in \mathcal{T}$ **do**
10 $\tau' \leftarrow \text{ExteND}(\tau, \pi_{i-1})$
11 $D \leftarrow D \cup \{\tau'_{n:n} \mid 1 \le n \le |\tau'|\}$
12 **if** $|\tau'| > t$ **then**
13 $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\tau'\}$
14 **else**
15 $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\tau\}$
16 $\pi_i \leftarrow \text{TrainVerifiablePolicy}(D)$
17 $\mathcal{T} \leftarrow \mathcal{T}'$
18 **return** SelectBest$(\{\pi_i \mid 1 \le i \le i_{\max}\})$

Algorithm 2: Subroutine ExteND

Input:
π : the current policy
$\tau = s_1 a_1 \cdots s_t a_t$: a path
Output: Maximal non-divergent prefix / extension of τ

1 $\tau' \leftarrow \varnothing$ /* an empty path */
2 **for** $n = 1, 2, 3, \ldots$ **do**
3 **if** $n > t$ **then**
4 sample $s_n \sim \mathcal{P}(s_{n-1}, a_{n-1})(\cdot)$
5 $a_n \leftarrow \pi^*(s_n)$
6 **else**
7 $s_n a_n \leftarrow \tau_{n:n}$
8 $\tau' \leftarrow \tau' s_n a_n$
9 **if** $\pi(s_n) \neq a_n \vee n$ is terminal index **then**
10 **break**
11 **return** τ'

learning algorithms, the prefixes have the same distribution as prefixes of paths generated by the oracle π^*.

Line 1 initializes the empty multiset $\mathcal{T}$ used to contain all the generated paths.[1] Paths are not necessarily unique, so a multiset is required to accurately represent the distribution. Lines 3 to 6 add the initial state-actions of k new paths to $\mathcal{T}$. Lines 7 and 8 initialize the multisets $\mathcal{T}'$—used to update $\mathcal{T}$—and D that stores the state-actions used to train the verifiable policy in line 16. We deliberately do not specify the nature of the subroutine TrainVerifiablePolicy, but note that our implementation trains a decision tree using the standard method provided by Scikit-Learn [18]. As in the case of paths, D is a multiset to accurately represent the distribution of state-actions. In line 10, a maximal non-divergent path τ' is calculated by the ExteND subroutine (Algorithm 2, described below) w.r.t. each path τ in $\mathcal{T}$ and the previous policy π_{i-1}. Here, maximal means a path of length $\min(\ell(\pi_{i-1}|\tau) + 1, T)$. The states of each extended path are added to the multiset D in line 11. Note that due to the uncertainties introduced by sampling and by training the verifiable policy, τ' may be shorter than τ. Hence, in lines 12 to 15, the longer of τ and τ' is added to the multiset $\mathcal{T}'$, which becomes the new $\mathcal{T}$ in line 17. $\mathcal{T}$ therefore grows monotonically. Finally, the policy with the greatest expected NPL is chosen by subroutine SelectBest in line 18.

[1] A set that allows multiple instances of elements.

SelectBest: We provide an algorithm for SelectBest in Appendix B and simply comment here that it refines the set of generated policies by iteratively estimating the expected NPL of each policy and keeping a subset of the policies with the highest estimates. The expected NPL is estimated by averaging the NPL of rolled out paths, using approximately the same total number of rollouts per iteration. Hence, as the set of policies is reduced, the accuracy of the estimates increases.

ExteND: (Algorithm 2) returns a maximal non-divergent path w.r.t. input path τ and policy π, where the maximal length is defined as $\min(\ell(\pi_{i-1}|\tau) + 1, T)$. By construction, τ is a path generated by the oracle, but its maximality w.r.t. an arbitrary policy π is initially unknown. Hence, the loop in line 2 to line 10 iterates from the initial state-action of τ until a disagreement or a terminal index is found (line 9). The output path τ' is constructed by copying state-actions from τ in line 8. In the case that the end of τ is reached without the discovery of a disagreement or a terminal index, new state-actions are sampled from the oracle in lines 4 and 5. In the case that the loop terminates with a disagreement, the disagreeing state-action is included in the returned path. This ensures that Algorithm 1 continues to explore.

A. Theoretical Aspects of NDI

The main idea of NDI lies in the equivalence between (2) and (5). That is, NDI at each iteration $i > 2$ approximately solves:

$$\pi_i \in \arg\max_{\pi \in \Pi} \mathbb{E}\big[\ell(\pi \mid \tau_{1:\ell_{i-1}+1})\big] \tag{6}$$

where $\ell_{i-1} := \ell(\pi_{i-1}|\tau)$ denotes the NPL of the last policy π_{i-1}. This iterative scheme *optimistically* estimates the NPL of π_i by "$\ell_{i-1} + 1$" from the last policy π_{i-1}. It expects

that the process (6) with slightly more data added due to "+1" will increase the NPL of π_i over π_{i-1}. This heuristic is also from the fact that maximizing (6) leads us to a higher expected NPL, thanks to the equivalence in Theorem 2, under a properly chosen path length that is in our case $\ell_{i-1} + 1$.

To analyze better our NDI, consider a fixed point $\pi_\bullet$ of (6) (assumed to exist) with its NPL $\ell_\bullet := \ell(\pi_\bullet|\tau)$. Being a fixed point of (6) means that $\pi_\bullet$ satisfies:

$$\pi_\bullet \in \arg\max_{\pi\in\Pi} \mathbb{E}\big[\ell(\pi \mid \tau_{1:\ell(\pi|\tau)+1})\big]. \tag{7}$$

Obviously, (7) is very similar to the problem (5), but is not the same, due to the existence of the "+1" in the length. If the path length were given by ℓ_{i-1}, rather than $\ell_{i-1}+1$, then the fixed point $\pi_\bullet$ would be the same as the solution $\hat{\pi}$. In this case, we might also expect a better convergence. However, we observe that a variant of NDI that has no "+1" heuristic produces DTs with expected NPL much lower than Algorithm 1. We hypothesize that this is caused by not sufficiently increasing the amount of data in D. In contrast, Algorithm 1 increases the paths in D by the minimal amount, i.e., "+1" state-action, which gives the following sub-optimality of $\pi_\bullet$:

1) If $\ell_\bullet$ is long enough, then $\ell_\bullet \approx \ell_\bullet + 1$, by which and (5) we can expect $\pi_\bullet \approx \hat{\pi}$. That is, the fixed point $\pi_\bullet$ of (6) can be still close to the solution $\hat{\pi}$ when $\ell_\bullet \gg 1$.
2) $\ell_\bullet + 1$ is the closest upper-bound of $\ell_\bullet$. I.e., (7) minimally violates the equivalence between the NPL ℓ and its surrogate ℓ in Theorem 2. Without such equivalence, minimizing (5) is not necessarily equal to optimizing the expected NPL (2) in general.

Intuitively, the "+1" heuristic minimally increases the amount of data to allow the NPL of each path to increase, while preserving the existing optimality as much as possible and not forgetting what it has already learned. We leave further theoretical analysis to future work.

VI. Experiments

In this section, we describe experiments that we conducted to demonstrate the performance of our NDI algorithm (Algorithm 1). We evaluate our method on DQN controllers for the CartPole [19], MountainCar [20], and Pong [21] reinforcement learning environments. For CartPole and Pong we used pre-trained DQNs provided by Keras-RL [22] and Viper,[2] respectively. We trained our own DQN for MountainCar, whose structure is loosely based on that of the DQN for CartPole.

CartPole and MountainCar are continuous space dynamical systems and their DQNs map Euclidean space to discrete actions. Pong's DQN maps images to actions, so we employ the abstraction used by Viper.[2] To train our verifiable policies, we use the standard CART method [23] in scikit-learn [18] to train decision trees using the Gini impurity measure.

Having previously established that other imitation learning algorithms used for verification are less effective than behavioural cloning (BC) (see, e.g., Fig. 1), we use BC as the baseline to judge the performance of Algorithm 1 w.r.t. expected NPL, model size, and training data efficiency. Each iteration of Algorithm 1 constructs a multiset of generated paths $\mathcal{T}$, a multiset of state-actions D, and a policy trained on D. For comparison, we train another policy using $|D|$ state-actions from complete paths generated by the oracle. These state-actions are not selected at random, but in the sequence that they are generated, up to $|D|$. We thus compare two sets of models: one trained on state-actions from non-divergent prefixes (denoted NDI in the figures) and one trained on a corresponding number of state-actions from complete paths (denoted BC in the figures). Note that the number of state-actions in the paths of $\mathcal{T}$ is what we record as the amount of generated data, which is greater than or equal to the amount of training data $|D|$.

To simplify comparisons, as far as possible we use the same parameters for all experiments and environments. In particular, we set $k = 25$ paths per iteration and $i_{\max} = 40$ maximum iterations for all experiments, relying on default settings in scikit-learn for training decision trees. An exception to this is that we set the maximum tree depth to 12 for Pong and CartPole, but set it to 5 for MountainCar. The value 12 is inherited from the implementation of Viper,[2] but this is excessive for MountainCar, which has a simpler controller.

For each environment, we performed 50 independent experiments, each comprising an execution of Algorithm 1 and a matched iterative procedure for behavioural cloning, as described above. In the following figures, we plot per-iteration summary statistics w.r.t. the 50 experiments. In most cases we plot the mean and quartiles, but in the case of MountainCar and Pong, the distributions of expected NPL for behavioural cloning are bimodal and skewed, making the upper quartile unstable or uninformative. In these cases, we therefore include the standard deviation as a more meaningful statistic.

A. MountainCar

The results for MountainCar are presented in Fig. 2.

In Fig. 2a we plot the expected NPL for models produced by NDI and BC. NDI clearly outperforms BC and its superiority increases with the number of iterations.

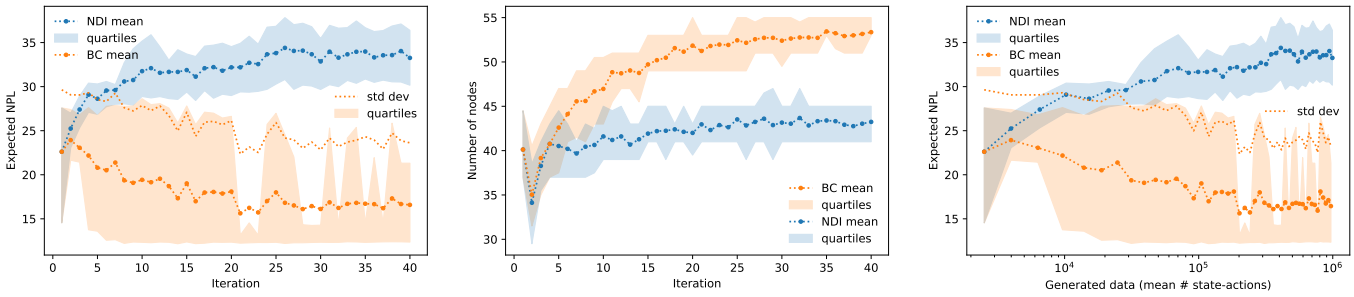
In Fig. 2b we plot the number of nodes in models produced at each iteration. As the performance w.r.t. NPL of the NDI models increase, as seen in Fig. 2a, the number of nodes increases, but the superiority of NDI over BC is once again clear. The number of nodes in models produced by BC initially increases much quicker and continues to rise, despite BC's performance w.r.t. NPL consistently decreasing.

In Fig. 2c, for each model we plot expected NPL against the amount of generated data. In contrast to NDI, we assume the amount of generated data for BC is the same as the amount of training data, since there is no selection. Despite this advantage, we see that NDI requires significantly less data to achieve its results than BC.

B. CartPole

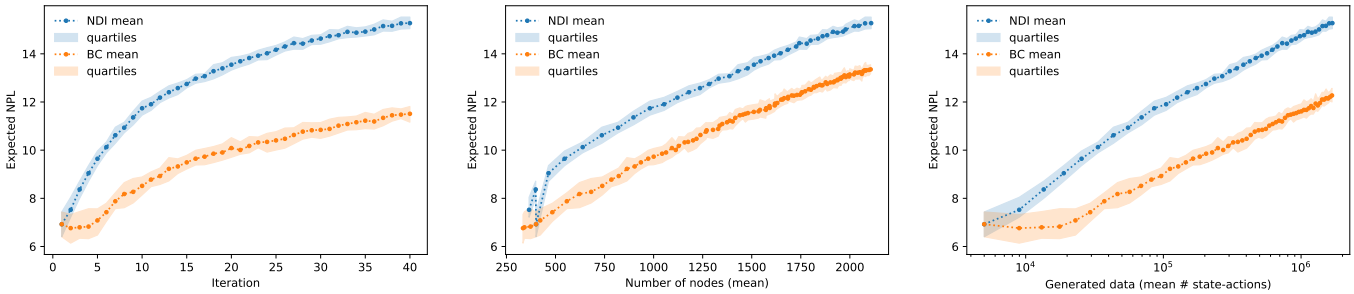
The CartPole problem has similarities with MountainCar, but its dynamics are more divergent near the initial states.

[2]https://github.com/obastani/viper



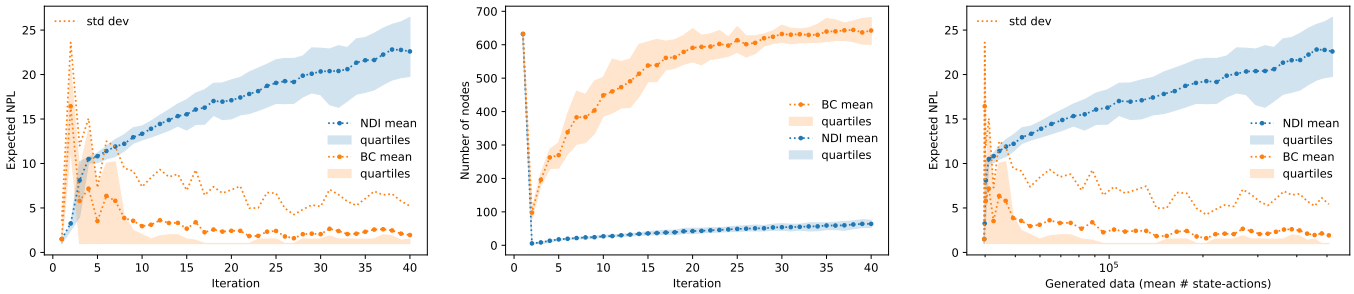
(a) Expected NPL of models (b) Number of nodes in DTs (c) Expected NPL vs generated data

Fig. 2: Results for MountainCar. Mean and quartiles w.r.t. 50 experiments. Expected NPL estimated with 10000 rollouts.



(a) Expected NPL of models (b) Expected NPL vs # of nodes in DTs (c) Expected NPL vs generated data

Fig. 3: Results for CartPole. Mean and quartiles w.r.t. 50 experiments. Expected NPL estimated with 10000 rollouts.



(a) Expected NPL of models (b) Number of nodes in DTs (c) Expected NPL vs generated data

Fig. 4: Results for Pong. Mean and quartiles w.r.t. 50 experiments. Expected NPL estimated with 20000 rollouts.

This is reflected in our results, which are presented in Fig. 3.

In Fig. 3a we plot the expected NPL of each model produced by NDI and BC. The superiority of NDI is evident, but the two curves are less divergent than those in Fig. 2a. We note that while the relative performance of NDI increases with iterations, the performance of BC also rises.

As a result of what we see in Fig. 3a, the superiority of NDI in terms of model size is also more subtle than in Fig. 2b. Hence, to demonstrate that NDI nevertheless produces more compact models than BC, in Fig. 3b we plot expected NPL against number of nodes for all the models. The curve for NDI clearly dominates that of BC.

Fig. 3c plots expected NPL against the amount of generated data. Once again, NDI clearly dominates BC, but in a less striking way than in Fig. 2c.

C. Pong

The results for Pong are presented in Fig. 4.

Pong poses a unique challenge for which NDI seems particularly effective. The original game is based on images, making its input space and dynamics out of scope for the type of verification we consider. To include it as an example in our work, we therefore adopt the transformation used by Viper[2] for the same purpose. This non-conservative abstraction

constructs a vector of seven integer features from a sequence of four $210 \times 160 \times 3$ image frames. The resulting loss of information results in conflicts in the training data (aleatoric uncertainty), such that different game states map to the same abstract state. While this does not necessarily prevent learning an adequate controller to play the game, as shown in [4], it reduces the maximum possible behavioural fidelity w.r.t. the oracle.

Fig. 4a demonstrates that NDI dramatically outperforms BC in terms of expected NPL. With increasing iterations, BC appears to converge to near zero NPL, while NDI continues to rise, with no apparent plateau within 40 iterations. The robust peak for BC at iteration 2 reflects the fact that the training data comes from just one short prefix, i.e., $|D| \approx 133 \ll$ the length of a complete path (ca 1600) and therefore has few conflicts.

The numbers of nodes plotted in Fig. 4b show that NDI's models are small and increase in size in line with NPL. By contrast, the size of BC's models increases rapidly, with a trend that is apparently inversely related to NPL. Complete paths of Pong are long, resulting in complex models. This explains the joint peak of NDI and BC at iteration 1, since both algorithms are trained on the same complete paths. After that, however, NDI only trains on non-divergent prefixes, while BC continues to train on long paths with conflicts that limit NPL.

As expected, the plot of expected NPL vs. generated data in Fig. 4c also shows the dominance of NDI over BC. NDI's performance here reflects the fact that it does not wastefully generate long paths with conflicts.

VII. Conclusion

Verification of learned controllers is an important challenge that is often difficult to address directly. In this work, we have focused on the idea of distilling a complex controller into a more tractable, verifiable model. The fidelity of the distilled model is crucial to the validity of the verification, so we have defined the notion of non-divergent path length (NPL) as a metric. Having found that existing distillation methods used for verification do not maintain good fidelity w.r.t. NPL, we proposed a non-divergent imitation algorithm (NDI) that gives significant improvements in performance. Our experiments distilling benchmark DQNs demonstrate that NDI typically achieves greater expected NPL, produces more compact models and is more data efficient than competing approaches. Our results also reveal some potentially unavoidable challenges with any distillation-based approach. Along with the usual requirement of having enough training data for the complexity of the problem, i.e., epistemic uncertainty, aleatoric uncertainty in the training data and lack of expressivity in the class of distilled models may bound the maximum fidelity possible. These challenges are the subject of our ongoing research.

References

[1] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *Computer Aided Verification*, 2017, pp. 97–117.

[2] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, *Algorithms for verifying deep neural networks*, 2020. arXiv: 1903.06758.

[3] D. Xu, D. Shriver, M. B. Dwyer, and S. Elbaum, "Systematic generation of diverse benchmarks for DNN verification," in *Computer Aided Verification*, 2020, pp. 97–121.

[4] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *NeurIPS*, 2018, pp. 2494–2504.

[5] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*. Springer International Publishing, 2018, pp. 1–1210.

[6] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, 2011, pp. 627–635.

[7] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[8] S. Ross and J. A. Bagnell, "Efficient reductions for imitation learning," in *Journal of Machine Learning Research*, vol. 9, 2010.

[9] X. Zhang, Y. Li, Z. Zhang, and Z.-L. Zhang, "f-gail: Learning f-divergence for generative adversarial imitation learning," in *NeurIPS*, 2020.

[10] P. Barde, J. Roy, W. Jeon, *et al.*, "Adversarial soft advantage fitting: Imitation learning without policy optimization," in *NeurIPS*, 2020.

[11] M. Vasic, A. Petrovic, K. Wang, *et al.*, *Moet: Interpretable and verifiable reinforcement learning via mixture of expert trees*, 2019. arXiv: 1906.06717.

[12] A. Verma, H. Le, Y. Yue, and S. Chaudhuri, "Imitation-projected programmatic reinforcement learning," in *NeurIPS*, vol. 32, 2019, pp. 15 752–15 763.

[13] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan, "An inductive synthesis framework for verifiable reinforcement learning," in *PLDI*, 2019, pp. 686–701.

[14] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, S. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," in *AAAI*, 2018.

[15] A. Jhunjhunwala, J. Lee, S. Sedwards, V. Abdelzad, and K. Czarnecki, "Improved policy extraction via online Q-value distillation," in *IJCNN*, 2020, pp. 1–8.

[16] J. Törnblom and S. Nadjm-Tehrani, "Formal verification of input-output mappings of tree ensembles," *Science of Computer Programming*, vol. 194, 2020.

[17] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018.

[20] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-209, Nov. 1990.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[22] M. Plappert, *Keras-rl*, https://github.com/keras-rl/keras-rl, 2016.

[23] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

Appendix A
Proof of Theorem 1 (Equivalence)

In this appendix we provide a proof of *equivalence* in Theorem 1. Let $\mathbb{P}^{\pi}[\cdot]$ and $\mathbb{P}^{\pi'}[\cdot]$ be the probability measures induced by the respective policies π and π'. Then,

$$\mathbb{P}^{\pi'}[\pi(s_1) = a_1] = \sum_{s \in \mathcal{S}, \pi(s) = \pi'(s)} d(s) = \mathbb{P}^{\pi}[\pi'(s_1) = a_1],$$

where $s_1 \sim d(\cdot)$. Next, we suppose that for $t \in \{1, \cdots, T-1\}$,

$$\mathbb{P}^{\pi}\left[\pi'(s_{1:t}) = a_{1:t}\right] = \mathbb{P}^{\pi'}\left[\pi(s_{1:t}) = a_{1:t}\right] \tag{8}$$

and prove $\mathbb{P}^{\pi}\left[\pi'(s_{1:t+1}) = a_{1:t+1}\right] = \mathbb{P}^{\pi'}\left[\pi(s_{1:t+1}) = a_{1:t+1}\right]$. The probabilities are trivially zeros if so are those in (8), by

$\{\pi'(s_{1:t+1}) = a_{1:t+1}\} \subseteq \{\pi'(s_{1:t}) = a_{1:t}\}$ and monotonicity. Otherwise, by definition of conditional probability and (8),

$$\begin{aligned}
&\mathbb{P}^{\pi'}\big[\pi(s_{1:t+1}) = a_{1:t+1}\big] \\
&= \mathbb{P}^{\pi'}\big[\pi(s_{t+1}) = a_{t+1} \,\big|\, \pi(s_{1:t}) = a_{1:t}\big] \cdot \mathbb{P}^{\pi'}\big[\pi(s_{1:t}) = a_{1:t}\big] \\
&= \mathbb{P}^{\pi'}\big[\pi(s_{t+1}) = \pi'(s_{t+1}) \,\big|\, \pi(s_{1:t}) = a_{1:t}\big] \cdot \mathbb{P}^{\pi}\big[\pi'(s_{1:t}) = a_{1:t}\big]
\end{aligned}$$

(under $\pi'(s_{1:t}) = a_{1:t} = \pi(s_{1:t})$, the states $s_{1:t+1}$ generated by π' has the same distributions as those generated by π hence)

$$\begin{aligned}
&= \mathbb{P}^{\pi}\big[\pi(s_{t+1}) = \pi'(s_{t+1}) \,\big|\, \pi'(s_{1:t}) = a_{1:t}\big] \cdot \mathbb{P}^{\pi}\big[\pi'(s_{1:t}) = a_{1:t}\big] \\
&= \mathbb{P}^{\pi}\big[\pi'(s_{t+1}) = a_{t+1} \,\big|\, \pi'(s_{1:t}) = a_{1:t}\big] \cdot \mathbb{P}^{\pi}\big[\pi'(s_{1:t}) = a_{1:t}\big] \\
&= \mathbb{P}^{\pi}\big[\pi'(s_{1:t+1}) = a_{1:t+1}\big].
\end{aligned}$$

Therefore, (8) is true for all $t = 1, \cdots, T$ and its application to the expected NPL finally yields

$$\begin{aligned}
\mathbb{E}\big[\mathit{l}(\pi|\tau^{\pi'})\big] &= \textstyle\sum_{t=1}^{T} \mathbb{P}^{\pi'}\big[\pi(s_{1:t}) = a_{1:t}\big] \\
&= \textstyle\sum_{t=1}^{T} \mathbb{P}^{\pi}\big[\pi'(s_{1:t}) = a_{1:t}\big] = \mathbb{E}\big[\mathit{l}(\pi'|\tau^{\pi})\big]
\end{aligned}$$

when the terminal index T is constant (i.e., determined only by timeout).

In general, T is a stopping time, i.e., each event $\{T \leq t\}$ depends at most on the states and actions $s_1 a_1 \cdots s_t a_t$ up to instant t. To complete the proof with this general case, note first that an NPL (1) for a complete path τ satisfies

$$\mathit{l}(\pi|\tau) = \textstyle\sum_{t=1}^{N} \mathbf{1}\big[\pi(s_{1:t}) = a_{1:t} \text{ and } t \leq T\big], \tag{9}$$

where $N \in \mathbb{N}$ denotes the timeout thus satisfies $T \leq N$. Here, the conditions inside the indicator function satisfy:

$$\begin{aligned}
&\mathbb{P}^{\pi'}\big[t \leq T \,|\, a_{1:t} = \pi(s_{1:t})\big] = \mathbb{P}^{\pi'}\big[t \leq T \,|\, \pi'(s_{1:t}) = \pi(s_{1:t})\big] \\
&= 1 - \mathbb{P}^{\pi'}\big[T \leq t-1 \,|\, \pi'(s_{1:t}) = \pi(s_{1:t})\big]
\end{aligned}$$

($\{T \leq t-1\}$ depends at most on $s_1 a_1 \cdots s_{t-1} a_{t-1}$ and under $\pi'(s_{1:t}) = a_{1:t} = \pi(s_{1:t})$, those states (and actions) generated by π' have the same distributions as those generated by π, so)

$$\begin{aligned}
&= 1 - \mathbb{P}^{\pi}\big[T \leq t-1 \,|\, \pi'(s_{1:t}) = \pi(s_{1:t})\big] \\
&= \mathbb{P}^{\pi}\big[t \leq T \,|\pi'(s_{1:t}) = \pi(s_{1:t})\big] = \mathbb{P}^{\pi}\big[t \leq T \,|\pi'(s_{1:t}) = a_{1:t}\big].
\end{aligned}$$

Employing this equality and (8), we obtain

$$\begin{aligned}
&\mathbb{P}^{\pi'}\big[\pi(s_{1:t}) = a_{1:t} \text{ and } t \leq T\big] \\
&\quad= \mathbb{P}^{\pi'}\big[t \leq T \,|\, \pi(s_{1:t}) = a_{1:t}\big] \cdot \mathbb{P}^{\pi'}\big[\pi(s_{1:t}) = a_{1:t}\big] \\
&\quad= \mathbb{P}^{\pi}\big[t \leq T \,|\, \pi'(s_{1:t}) = a_{1:t}\big] \cdot \mathbb{P}^{\pi}\big[\pi'(s_{1:t}) = a_{1:t}\big] \\
&\quad= \mathbb{P}^{\pi}\big[\pi'(s_{1:t}) = a_{1:t} \text{ and } t \leq T\big],
\end{aligned}$$

provided that $\mathbb{P}^{\pi}[\pi(s_{1:t}) = a_{1:t}] \neq 0$ (otherwise, by monotonicity with $\{\pi(s_{1:t}) = a_{1:t} \text{ and } t \leq T\} \subseteq \{\pi(s_{1:t}) = a_{1:t}\}$ and (8), those probabilities are also zeros). Therefore, applying the last equality to (9), we finally have

$$\begin{aligned}
\mathbb{E}\big[\mathit{l}(\pi|\tau^{\pi'})\big] &= \textstyle\sum_{t=1}^{N} \mathbb{P}^{\pi'}\big[\pi(s_{1:t}) = a_{1:t} \text{ and } t \leq T\big] \\
&= \textstyle\sum_{t=1}^{N} \mathbb{P}^{\pi}\big[\pi'(s_{1:t}) = a_{1:t} \text{ and } t \leq T\big] \\
&= \mathbb{E}\big[\mathit{l}(\pi'|\tau^{\pi})\big]
\end{aligned}$$

(where the timeout N is constant, but T is not in general) and the proof of equivalence in Theorem 1 is now completed.

Appendix B
Subroutine to Select Best Policy

We assume an MDP $(\mathcal{S}, \mathcal{A}, d, \mathcal{P})$ and an oracle π^*, as described in Section III. The outer loop (lines 3 to 21) iteratively refines the input set of policies by repeatedly selecting the half of the current set with the highest estimated expected NPL. The loop terminates when there is just one remaining policy, which is returned in line 22. For each of the current policies (lines 5 to 18), lines 7 to 17 calculate the NPL of $i_{\max}$ rolled-out paths. Line 19 creates a bijection to order the current polices w.r.t. estimated expected NPL, which line 20 uses to select the upper 50^{th} percentile. This refinement approximately halves the size of M', depending on whether $|M'|$ is odd or even, while line 21 doubles the number of rollouts per policy, thus approximately maintaining the same total number of rollouts per iteration and increasing the accuracy of subsequent estimations.

Algorithm 3: Subroutine SelectBest

Input: $\mathcal{M}$: a set of candidate policies
Output: A policy $\pi \in \mathcal{M}$ with greatest estimated expected NPL

1 $M \leftarrow \mathcal{M}$
2 $i_{\max} \leftarrow$ minimum number of rollouts per policy
3 **while** $|M| > 1$ **do**
4 $M' \leftarrow \varnothing$
5 **foreach** $\pi \in M$ **do**
6 $l \leftarrow 0$
7 **repeat** $i_{\max}$ **times**
8 **for** $t = 1, 2, \ldots$ **do**
9 **if** $t = 1$ **then**
10 sample $s_t \sim d(\cdot)$
11 **else if** $t - 1$ is terminal index **then**
12 **break**
13 **else**
14 sample $s_t \sim \mathcal{P}(s_{t-1}, \pi^*(s_{t-1}))(\cdot)$
15 **if** $\pi(s_t) \neq \pi^*(s_t)$ **then**
16 **break**
17 $l \leftarrow l + t/i_{\max}$
18 $M' \leftarrow M' \cup \{(\pi, l)\}$
19 Let ϕ: $M' \leftrightarrow \{1, 2, \ldots, |M'|\}$ s.t. $\forall (\pi, l), (\pi', l') \in M'$ $\phi((\pi, l)) > \phi((\pi', l')) \iff l > l'$
20 $M \leftarrow \{\pi \mid (\pi, l) \in M' \text{ and } \phi((\pi, l)) \geq \lceil |M'|/2 \rceil\}$
21 $i_{\max} \leftarrow 2 \cdot i_{\max}$
22 **return** $\pi \mid \exists! \pi \in M$