Scaling Safe Policy Improvement: Monte Carlo Tree Search and Policy Iteration Strategies

FEDERICO BIANCHI*, University of Verona, Italy
ALBERTO CASTELLINI, University of Verona, Italy
EDOARDO ZORZI, University of Verona, Italy
THIAGO D. SIMÃO, Eindhoven University of Technology, Netherlands
MATTHIJS T. J. SPAAN, Delft University of Technology, Netherlands
ALESSANDRO FARINELLI, University of Verona, Italy

Offline Reinforcement Learning (RL) allows policies to be trained on pre-collected datasets without requiring additional interactions with the environment. This approach bypasses the need for real-time data acquisition in real-world applications, which can be impractical due to the safety issues inherent in the learning process. However, offline RL faces significant challenges, such as distributional shifts and extrapolation errors, and the resulting policies might underperform compared to the baseline policy. Safe policy improvement algorithms mitigate these issues, enabling the reliable deployment of RL approaches in real-world scenarios where historical data is available, guaranteeing that any policy changes will not result in worse performance compared to the baseline policy used to collect training data. In this paper, we propose MCTS-SPIBB, an algorithm that leverages Monte Carlo Tree Search (MCTS) for scaling safe policy improvement to large domains. We theoretically prove that the policy generated by MCTS-SPIBB converges to the optimal safely improved policy produced by Safe Policy Improvement with Baseline Bootstrapping (SPIBB) as the number of simulations increases. Additionally, we introduce SDP-SPIBB, a novel extension of SPIBB designed to address the scalability limitations of the standard algorithm via Scalable Dynamic Programming. Our empirical analysis across four benchmark domains demonstrates that MCTS-SPIBB and SDP-SPIBB significantly enhance the scalability of safe policy improvement, providing robust and efficient algorithms for large-scale applications. These contributions represent a significant step towards the deployment of safe RL algorithms in complex real-world environments.

JAIR Associate Editor: Scott Sanner

JAIR Reference Format:
Federico Bianchi, Alberto Castellini, Edoardo Zorzi, Thiago D. Simão, Matthijs T. J. Spaan, and Alessandro Farinelli. 2025. Scaling Safe Policy Improvement: Monte Carlo Tree Search and Policy Iteration Strategies. *Journal of Artificial Intelligence Research* 84, Article 22 (November 2025), 55 pages. DOI: 10.1613/jair.1.19649

*Corresponding Author.

Authors' Contact Information: Federico Bianchi, ORCID: 0000-0001-7773-3032, federico.bianchi@univr.it, University of Verona, Verona, Italy; Alberto Castellini, ORCID: 0000-0001-8420-0699, alberto.castellini@univr.it, University of Verona, Verona, Italy; Edoardo Zorzi, ORCID: 0000-0001-7090-9592, edoardo.zorzi@univr.it, University of Verona, Verona, Italy; Thiago D. Simão, ORCID: 0000-0002-3568-9464, t.simao@tue.nl, Eindhoven University of Technology, Eindhoven, Netherlands; Matthijs T. J. Spaan, ORCID: 0009-0002-2858-8611, M.T.J.Spaan@tudelft.nl, Delft University of Technology, Delft, Netherlands; Alessandro Farinelli, ORCID: 0000-0002-2592-5814, alessandro.farinelli@univr.it, University of Verona, Verona, Italy.

DOI: 10.1613/jair.1.19649

1 Introduction

Safety is a paramount requirement for deploying Reinforcement Learning in real-world scenarios (Sutton and Barto 2018). Safe RL (García and Fernández 2015) addresses these concerns by learning policies that not only maximize expected returns but also ensure minimal error levels or satisfy safety constraints during learning. Safe RL focuses on incorporating these constraints directly into the learning process to prevent the agent from taking harmful actions. This is especially important in critical applications such as autonomous driving, healthcare, and robotics, where the reliability of control policies is a crucial issue. However, in such real-world applications, collecting experience can be impractical due to the safety issues inherent in the learning process (Bonanni et al. 2025; Mazzi, Castellini, et al. 2023; Zorzi et al. 2025). A core foothold towards the deployment of RL applications in real-world domains is the ability to reliably make good decisions with minimum interactions with the environment since data collection is expensive, risky, or impractical (Dulac-Arnold et al. 2021). Offline RL (Levine et al. 2020; Prudencio et al. 2022) addresses these issues by complementing Safe RL, allowing policies to be trained on pre-collected datasets without requiring additional interactions with the environment. Offline RL leverages existing data to train models that generalize well to new situations, thus bypassing the need for real-time data acquisition. However, offline RL faces significant challenges, such as distributional shifts and extrapolation errors (Fujimoto et al. 2019; Kumar, Fu, et al. 2019), which occur when the policy encounters states and actions that are not well-represented in the training data. These challenges can make offline RL algorithms unreliable, as the computed policy might underperform compared to the baseline policy.

To mitigate these risks, Safe Policy Improvement (SPI) (Thomas et al. 2015) techniques are crucial and guarantee that any policy changes will not result in worse performance compared to the baseline policy used to collect training data, thus addressing the reliability issues inherent in offline RL. SPI considers a percentile criterion (Delage and Mannor 2010) to optimize the policy in the worst-case scenario and can be classified into two main categories based on how they utilize the uncertainty of state-action pairs (Scholl et al. 2022a): i) methods that apply uncertainty to the action-value function by reducing the value of uncertain actions, thereby adapting the Policy Evaluation step; and ii) methods that apply uncertainty to restrict the set of policies, by limiting the set of policies that can be learned, thereby adapting the policy iteration step. Safe Policy Improvement with Baseline Bootstrapping (Laroche et al. 2019) is a state-of-the-art algorithm for SPI that constrains the policy space and extends policy iteration by bootstrapping the baseline policy. This strategy effectively constrains the search for an improved policy within a space where the estimate of the model is sufficiently precise. While SPIBB represents a significant advancement in ensuring safe policy improvements, its applicability is limited by computational complexity, especially in large domains.

SPIBB and other state-of-the-art SPI algorithms (Scholl et al. 2022a) have been investigated mostly as extensions of the policy iteration algorithm, which is inherently computationally intensive. Policy iteration involves two main steps that are computationally demanding, namely, Policy Evaluation and Policy Improvement. The complexity of these steps increases significantly with the size of the state space because all states are considered in the iterative evaluation and improvement processes. Specifically, the computational cost increases as the state space and action space expand. When a matrix-based approach is used for Policy Evaluation, it requires solving a system of linear equations and performing matrix inversion, which typically has a time complexity of $O(|S \times A|^3)$. Alternatively, if a dynamic programming approach is adopted, the time complexity is $O(|S|^2|A|^2)$, where $|S|$ is the number of states and $|A|$ is the number of actions. Given the quadratic or cubic scaling of these operations, SPIBB algorithms become impractical for large state spaces. In addition to the time complexity problem, there is also the problem of memory complexity, since the memory needed to store and process value functions and policies increases significantly with the size of the state space. To address the scalability issues of SPIBB, we propose a novel approach and an extension of SPIBB.

The first approach proposed in this work, called MCTS-SPIBB, improves the scalability of SPIBB by computing a policy in an online fashion and focusing only on reachable states. MCTS-SPIBB is a sampling-based method that integrates Monte Carlo Tree Search (Browne et al. 2012; Coulom 2007) with SPI. MCTS is an online method that efficiently computes policies through a look-ahead search, making it suitable for large state spaces. However, incorporating MCTS within the context of SPI presents challenges because MCTS operates online, meaning it does not compute policies for all states. Additionally, MCTS performs simulations to approximate state-action values by considering future rewards, a process not present in SPIBB, which is based on the policy improvement paradigm. MCTS-SPIBB addresses these challenges by using MCTS as a solution strategy to achieve a safe improvement of the baseline policy with much better scaling properties than SPIBB.

The second approach proposed in this paper, called Scalable Dynamic Programming SPIBB (SDP-SPIBB), is a novel extension of SPIBB designed to improve its scalability by accelerating the dynamic programming approach and skipping unnecessary computational steps (Bianchi et al. 2025). SDP-SPIBB efficiently handles large domains by using strategies that restrict the Policy Evaluation and Policy Improvement to state-action sub-spaces where computation is feasible, thereby addressing the limitations of the standard SPIBB algorithm. This extension enables the application of SDP-SPIBB in environments that were previously infeasible to handle with the standard SPIBB algorithm due to computational constraints. The empirical analysis considers the performance of MCTS-SPIBB and SDP-SPIBB across several domains, with a focus on their scalability and safety. The results show that both MCTS-SPIBB and SDP-SPIBB scale effectively to domains larger than those manageable by SPIBB and state-of-the-art SPI algorithms while maintaining safety guarantees.

In summary, this work proposes five contributions[1]: *i)* a formalization of the problem of SPI in the MCTS framework; *ii)* an online sampling-based algorithm for SPI on MDPs, called MCTS-SPIBB, with complexity independent from state space, action space, and dataset dimension; *iii)* a theoretical proof that the policy generated by MCTS-SPIBB converges asymptotically to the SPIBB policy, as the number of simulations increases; *iv)* a scalable version of SPIBB based on policy iteration, called SDP-SPIBB, with complexity related to the size of the dataset instead of the size of the state and action spaces; *v)* an extensive empirical evaluation of the scalability and safety of MCTS-SPIBB and SDP-SPIBB on both single and multi-agent domains. Improving the scalability of SPI algorithms opens new possibilities for their application in large-scale or real-world environments. This advancement broadens the scope of problems that can be tackled reliably with RL. By addressing the computational limitations, our proposed methods contribute to making safe policy improvement techniques more practical and effective in large-scale scenarios, ultimately trying to result in more intelligent and safer decision-making processes.

2 Background

In this section, we present the mathematical notations used in the next sections.

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) (Puterman 2014) are a mathematical framework used to model decision-making problems. An MDP is formally defined as a tuple $M = \langle S, A, T, R, \gamma \rangle$. In this representation, S is the set of states that represent all possible situations in the environment; A is the set of actions available; $T : S \times A \longrightarrow \mathcal{P}(S)$ is a probability distribution function that specifies the probability of transitioning from one state to another given an action, where $\mathcal{P}(X)$ represents the set of probability distributions over a finite set X; $R : S \times A \longrightarrow [-R_{max}, R_{max}]$ is a bounded stochastic reward function that provides a numerical reward received after transitioning from one state to another due to action and $\gamma \in [0, 1]$ is a discount factor used to discount future rewards, reflecting the

[1]This paper is an extension of our work presented to the International Conference on Machine Learning (ICML 2023) (Castellini et al. 2023), and the contributions listed as i), ii), and iii) below were also presented in that work.

preference for immediate rewards over future rewards. The goal of an MDP is to find a policy π, which is a mapping from states to actions, that maximizes the expected cumulative reward over time. The value function $V^\pi(s)$ under a policy π is defined as the expected return starting from state s and following policy π:

$$V_M^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s\right]. \tag{1}$$

The optimal value function $V^*(s)$ gives the maximum expected return achievable from state s:

$$V^*(s) = \max_\pi V^\pi(s). \tag{2}$$

Similarly, the action-value function $Q^\pi(s, a)$ is defined as the expected return starting from state s, taking action a, and thereafter following policy π:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a\right]. \tag{3}$$

The optimal action-value function $Q^*(s, a)$ is given by:

$$Q^*(s, a) = \max_\pi Q^\pi(s, a). \tag{4}$$

There are two main approaches for solving MDPs (Sutton and Barto 2018), exact methods, which compute the optimal policy considering all possible states and actions, and approximate methods, which compute near-optimal policies by approximating the value function. Exact methods for solving MDPs (Bellman 1957; Howard 1960; Sutton and Barto 2018) provide optimal policies that guarantee finding the highest possible expected cumulative reward. Common exact methods include dynamic programming approaches such as Value Iteration (Bellman 1957), and policy iteration (Howard 1960). The main limitation of exact methods lies in their computational complexity (Papadimitriou and J. Tsitsiklis 1987). As the size of the state and action spaces increases, the computational time and memory requirements grow, making exact methods impractical for large-scale or real-world problems. Additionally, exact methods require a complete and accurate model of the environment, which may not always be available or feasible to construct.

Approximate methods for solving MDPs provide scalable solutions for large-scale problems by using function approximators, random sampling, or direct interaction with the environment (Bertsekas and J. N. Tsitsiklis 1996; Silver and Veness 2010; Sutton and Barto 2018). Common approximate methods include Approximate Dynamic Programming (Kochenderfer et al. 2022), Proximal Policy Optimization (PPO) (Schulman et al. 2017), Deep Q-Network (DQN) (Mnih et al. 2013) and Monte Carlo Tree Search (Browne et al. 2012; Coulom 2007; Kocsis and Szepesvári 2006). These methods are flexible and can adapt to dynamic environments where the model is not fully known. However, approximate methods do not guarantee optimal solutions in general; the policies they produce are sub-optimal but may not achieve the highest possible expected reward. Despite these limitations, approximate methods are essential for problems where exact solutions are computationally prohibitive.

2.2 Monte Carlo Tree Search

Monte Carlo Tree Search is an online algorithm used in decision-making processes (Browne et al. 2012) that efficiently computes action values in a sample-based manner for a given state in which the agent is located. The algorithm builds a search tree from the initial state, where each node represents a state of the environment, and the directed links to child nodes represent actions leading to subsequent states. MCTS assumes that the transition function is known and employs techniques for optimally balancing exploration and exploitation (Kocsis and Szepesvári 2006), with the primary goal of identifying the most promising action by using simulations to estimate the value of different actions. It has proven highly successful in various applications, including board games like Go and Chess, as well as in complex video games (Vinyals et al. 2019). The MCTS algorithm begins by creating a root node, which represents the initial state. The algorithm then iteratively performs four main phases: *Selection*, *Expansion*, *Simulation*, and *Backpropagation.* These phases are executed repeatedly until a predefined computational budget, such as a time limit or several iterations, is exhausted.

During the selection phase, the algorithm begins at the root node and recursively selects child nodes until it reaches a leaf node. This selection process is guided by a tree policy that aims to balance exploration (selecting less-visited nodes) and exploitation (selecting nodes with higher rewards). A widely used method for this phase is the Upper Confidence Bound for Trees (UCT) (Kocsis and Szepesvári 2006), which selects the child node i that maximizes the following value:

$$UCT_i = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}, \tag{5}$$

where w_i is the total reward obtained through node i, n_i is the number of times node i has been visited, N is the total number of visits to the parent node, c is an exploration constant that controls the exploration-exploitation trade-off.

In the expansion phase, if the selected leaf node is not terminal, the expansion phase adds one or more child nodes to the tree, representing possible subsequent actions. This step allows the algorithm to explore new states and potential actions. In the simulation phase, a rollout is conducted from the newly expanded node. This involves simulating the entire run to the end from the current state, typically using a default (often random) policy. The outcome of this simulation provides an estimate of the value of the state. Finally, the backpropagation phase updates the values of the nodes along the path from the newly expanded node back to the root. This involves updating the visit counts and rewards for each node on the path. The backpropagated values help to inform future decisions in the selection phase.

2.3 Safe Policy Improvement

Let an unknown finite MDP $M^* = \langle S, A, T^*, R, \gamma \rangle$ represent a true environment where T^* is an unknown transition model and R a known reward function. $\Pi = \{S \longrightarrow \Delta_A\}$ is the set of stochastic policies, where Δ_A denotes the set of probability distributions over the set of actions A. Given a policy subset $\Pi' \subseteq \Pi$, a policy π' is Π'-optimal for an MDP M when it maximizes its performance on Π' (Laroche et al. 2019):

$$\rho(\pi', M) = \max_{\pi \in \Pi'} \rho(\pi, M), \tag{6}$$

where $\rho(\pi', M)$ represents the discounted cumulative return of policy π in the MDP M. SPI approaches focus on the Offline RL setting, where the algorithm does its best at learning a policy from a fixed set of experiences. Given a dataset $\mathcal{D} = (s_j, a_j, r_j, s'_j)_{j=1}^n$ collected by a baseline policy π_0, let $N_{\mathcal{D}}(s, a)$ denote the number of visits to the state-action pair $(s, a) \in \mathcal{D}$. We construct the Maximum Likelihood Estimator (MLE) $M^{\mathcal{D}} = \langle S, A, T^{\mathcal{D}}, R, \gamma \rangle$ of M^* as follows:

$$T^{\mathcal{D}}(s' \mid s, a) = \frac{\sum_{(s_j=s, a_j=a, s'_j=s') \in \mathcal{D}} \mathbb{1}}{N_{\mathcal{D}}(s, a)} \tag{7}$$

This is common in real-world domains where the transition model must be estimated or inferred from small amounts of data. The *safety* of the improvement must be guaranteed, specifically, π_I must outperform π_0 with an admissible performance loss. A significant approach in this context is the *percentile criterion* (Delage and Mannor 2010), which aims to improve policy performance while maintaining a high probability of improvement over a baseline policy π_0. The percentile criterion is defined as follows:

$$\pi_C = \arg\max_{\pi \in \Pi} \mathbb{E}[\rho(\pi, M) \mid M \sim \mathbb{P}_{MDP}(\cdot \mid \mathcal{D})], \tag{8}$$

subject to the constraint:

$$\mathbb{P}(\rho(\pi_I, M) \geq \rho(\pi_0, M) - \zeta \mid M \sim \mathbb{P}_{MDP}(\cdot \mid \mathcal{D})) \geq 1 - \delta, \tag{9}$$

where $\mathbb{P}_{MDP}(\cdot|\mathcal{D})$ represents the posterior probability distribution of the MDP parameters given the dataset $\mathcal{D}$, namely, the probability of the transition model $T^{\mathcal{D}}$ computed from the dataset $\mathcal{D}$, ρ denotes the policy performance, ζ is an approximation parameter (or precision level), and $1 - \delta$ denotes a high confidence level.

2.4 Safe Policy Improvement with Baseline Bootstrapping

The SPIBB (Laroche et al. 2019) algorithm reformulates the *percentile criterion* (see Equation 8) and aims to maximize the policy's performance in the estimated MDP $M^{\mathcal{D}}$, guaranteeing that the improved policy π_I is ζ-approximately at least as good as the baseline policy π_0:

$$\max_{\pi_I \in \Pi} \rho(\pi_I, M^{\mathcal{D}}), \text{ s.t.} \forall M \in \Xi,\ \rho(\pi_I, M) \geq \rho(\pi_0, M) - \zeta, \tag{10}$$

within the set of admissible MDPs Ξ:

$$\Xi(M^{\mathcal{D}}, e) = \{M \mid \forall (s,a) \in S \times A, \|T(s,a,\cdot) - T^{\mathcal{D}}(s,a,\cdot)\|_1 \leq e(s,a)\}, \tag{11}$$

where $e : S \times A \longrightarrow \mathbb{R}$ is an error function depending on $\mathcal{D}$ and δ. Based on Theorem 8 in (Petrik et al. 2016), SPIBB guarantees that if all state-action pair counts $N_{\mathcal{D}}(s,a)$ meet the condition:

$$N_{\mathcal{D}}(s,a) \geq N_{\wedge} = \frac{8V_{max}^2}{\zeta^2(1-\gamma)^2} \log\left(\frac{2|S||A|2^{|S|}}{\delta}\right), \tag{12}$$

and $M^{\mathcal{D}}$ is the Maximum Likelihood Estimation MDP, then with high probability $1 - \delta$, the optimal policy $\pi^* = \arg\max_{\pi \in \Pi} \rho(\pi, M^{\mathcal{D}})$ in $M^{\mathcal{D}}$ is ζ-approximately safe in the true environment M^*:

$$\rho(\pi^*, M^*) - \zeta \geq \rho(\pi_0, M^*) - \zeta. \tag{13}$$

Let $N_{\mathcal{D}}(s,a) \in \mathbb{N}$ be the number of occurrences of a state-action pair (s,a) in $\mathcal{D}$ and $N_{\wedge} \in \mathbb{N}$ a related threshold. SPIBB splits state-action pairs into two subsets: the bootstrapped subset

$$\mathcal{B} = \{(s,a) : N_{\mathcal{D}}(s,a) < N_{\wedge}\}, \tag{14}$$

which is the set of state-action pairs that occur less than $N_{\wedge}$ times in $\mathcal{D}$; the non-bootstrapped set

$$\overline{\mathcal{B}} = \{(s,a)\colon N_{\mathcal{D}}(s,a) \geq N_{\wedge}\}, \tag{15}$$

which is the set of state-action pairs that occur at least $N_{\wedge}$ times in $\mathcal{D}$. We also define for each state s, bootstrapped action sets

$$\mathcal{B}_A(s) = \{a \in A\colon (s,a) \in \mathcal{B}\}, \tag{16}$$

and non-bootstrapped action sets,

$$\overline{\mathcal{B}}_A(s) = \{a \in A\colon (s,a) \in \overline{\mathcal{B}}\}. \tag{17}$$

In practice, the baseline policy π_0 is executed in the true environment M^*, where a dataset is collected and then used to compute the estimated MDP $M^{\mathcal{D}}$. SPIBB computes policies that satisfy the Equation 10 and the search is done in the subspace of policies equal to the baseline in the state-action pairs not observed enough times in $\mathcal{D}$, namely,

$$\Pi_0 = \{\pi \in \Pi : \pi(s,a) = \pi_0(s,a) : \forall (s,a) \in \mathcal{B}\}. \tag{18}$$

While the SPIBB optimization (Equation 10) is carried out on the estimated MDP $M^{\mathcal{D}}$, the safety guarantee holds in the true environment M^*. Specifically, under the PAC framework formalized in (Laroche et al. 2019), if the count threshold condition $N_{\mathcal{D}}(s,a) \geq N_{\wedge}$ is satisfied for all relevant state-action pairs, then the resulting policy satisfies the performance bound stated in Equation 13, holding with high probability in the true environment M^*.

This result ensures that the improved policy will not degrade the baseline's performance beyond a tolerance ζ in the true environment, provided that updates are restricted to well-supported regions. The partitioning into bootstrapped and non-bootstrapped sets is therefore critical: it ensures that the improved policy deviates from the baseline only in regions where the model is well supported by data, thereby mitigating the effects of model bias. In this sense, SPIBB directly addresses a core challenge in offline RL safe generalization under model uncertainty by tying the scope of policy improvement to the confidence provided by empirical evidence.

2.4.1 SPIBB with Matrix Inversion (I-SPIBB). The first way to implement SPIBB is presented in Algorithm 1 and 2. This is the standard implementation released in (Laroche et al. 2019), and we refer to this implementation as I-SPIBB, where I-SPIBB stands for Matrix Inversion SPIBB. The algorithm performs policy evaluation by solving a system of linear equations, which requires a matrix inversion. The time complexity of I-SPIBB is $O(|S \times A|^3)$ due to the matrix inversion step, making it suitable for small state and action spaces but impractical for larger ones due to computational constraints.

Algorithm 1 I-SPIBB

```
Input: baseline policy $\pi_0$, transition matrix $T^{\mathcal{D}}$, reward matrix $R$, discount factor $\gamma$, stopping parameter $\epsilon$
Output: an improved policy $\pi$
1: $\pi = \pi_0$ // Initialization to the baseline policy
2: $Q = 0$
3: $\delta_Q = \infty$
4: while $\delta_Q > \epsilon$ do
5:     $M = I - \gamma E_{sum}(T^{\mathcal{D}}, \pi)$ // Einstein-summation + reshape
6:     $Q' = M^{-1}R$
7:     for $s$ in $S$ do
8:         $\pi(s)$ = Policy Improvement$(Q', \pi_0, s)$
9:     end for
10:    $\delta_Q = \|Q - Q'\|_\infty$
11:    $Q = Q'$
12: end while
```

Algorithm 2 Policy Improvement

```
Input: last iteration of $Q$, baseline policy $\pi_0$, state $s$
Output: policy improvement $\pi^I(s)$
1: Initialize $\pi^I(s) = 0$
2: for $a \in A | (s,a) \in \mathcal{B}$ do
3:     $\pi^I(s,a) = \pi_0(s,a)$
4: end for
5: $\pi^I(s, \operatorname{argmax}_{a|(s,a)\in\overline{\mathcal{B}}} Q(s,a)) = \sum_{a|(s,a)\in\overline{\mathcal{B}}} \pi_0(s,a)$
```

The algorithm begins by initializing the policy π to π_0, the Q-table to all zeros, and δ to a value greater than ϵ, where δ_Q represents the difference between the updated Q-values and the Q-values from the previous iteration (see Algorithm 1, lines 1-3). It then enters a loop (Algorithm 1, line 4) where it computes a matrix M as the difference between an identity matrix I and the result of the function E_{sum} (which involves an Einstein summation of $T^{\mathcal{D}}$ and π, followed by reshaping to generate a matrix of size $|S \times A| \times |S \times A|$). Subsequently, the Q-table is updated by multiplying the inverse of matrix M with the reward matrix R (Algorithm 1, lines 5-6). The algorithm proceeds to calculate policy improvements for each state $s \in S$ using the SPIBB constraints (as specified in Algorithm 2). It then updates the values of δ_Q and Q before starting a new iteration. The loop continues until δ_Q becomes greater than ϵ, when it terminates.

2.4.2 *SPIBB with Dynamic Programming (DP-SPIBB).*

A second possible implementation of SPIBB uses a dynamic programming approach for policy evaluation and avoids matrix inversion. We refer to this version as DP-SPIBB. The time complexity is $O(|S|^2|A|^2)$ due to its searching through the entire state and action space to find the optimal action. It scales better than I-SPIBB but remains computationally challenging for large state and action spaces. The pseudocode for DP-SPIBB (Algorithm 3) shows that the input and output are similar to I-SPIBB, with

Algorithm 3 DP-SPIBB

```
Input: baseline policy $\pi_0$, transition matrix $T^{\mathcal{D}}$, discount factor $\gamma$, stopping parameter $\epsilon$
Output: an improved policy $\pi$
1:  $\pi = \pi_0$ // Initialization of $\pi$ to $\pi_0$
2:  $Q = 0$
3:  $\delta_Q = \infty$
4:  while $\delta_Q > \epsilon$ do
5:      for $s$ in $S$ do
6:          for $a$ in $A$ do
7:              Initialize $\psi = 0$ // Return from the next states
8:              for $s'$ in $S$ do
9:                  for $a'$ in $A$ do
10:                     $\psi = \psi + Q(s', a')\pi(s', a')T^{\mathcal{D}}(s, a, s')$
11:                 end for
12:             end for
13:             $Q'(s, a) = R(s, a) + \gamma\psi$
14:         end for
15:     end for
16:     for $s$ in $S$ do
17:         $\pi(s)$ = Policy Improvement($Q', \pi_0, s$)
18:     end for
19:     $\delta_Q = \|Q - Q'\|_\infty$
20:     $Q = Q'$
21: end while
```

the difference that the reward matrix R can be defined as a function that takes a state-action pair (s, a) as input and returns a reward. The algorithm initializes the policy π to π_0, the Q-table to 0, and δ_Q to a value greater than ϵ (see Algorithm 3, lines 1-3). DP-SPIBB avoids matrix inversion by searching through the entire state space S and action space A to find the optimal action (Algorithm 3, lines 5-6). The return value ψ is initialized to zero, and a loop is initiated to iterate over the next reachable states, accumulating rewards from each state. Subsequently, it updates the Q-table for (s, a) based on the immediate reward obtained by calling the reward function and the discounted return from the next states (Algorithm 3, lines 7-11). The subsequent steps are equivalent to those of I-SPIBB, including the computation of policy improvements (see Algorithm 2) for each state $s \in S$ using the SPIBB constraints and the update of δ_Q and Q before initiating a new iteration.

3 Safe Policy Improvement via MCTS and Dynamic Programming

We first present the two scalable methods MCTS-SPIBB and SDP-SPIBB that represent the main contributions of the paper. For each method, we first provide an overview, then we describe the algorithm in depth, and subsequently, we analyze the complexity and the theoretical guarantees it provides. MCTS-SPIBB introduces MCTS into the SPIBB framework, making it applicable to domains with large state spaces, with a complexity depending on the number of simulations performed. SDP-SPIBB is based on the observation that the policy evaluation step of DP-SPIBB is inefficient, and its complexity can be reduced from a value depending on the full state-action space to one that scales with the size of the non-bootstrapped set, which is in turn related to the size of the trajectory dataset.

Both methods are model-based, and the true transition model is assumed to be unknown. Instead, it is learned via Maximum Likelihood Estimation from the offline dataset, following the standard practice in safe policy improvement literature, and used throughout MCTS-SPIBB or SDP-SPIBB. This MLE model forms the basis for policy improvement steps, and the bootstrapping mechanism ensures that policy updates remain restricted to well-supported state-action regions. Intuitively, policy improvement steps are constrained to state-action pairs that are sufficiently represented in the data, while in poorly supported state-action pairs, the baseline is preserved. This mitigates the risk introduced by model bias. Offline RL methods in the literature typically do not provide explicit safety guarantees on policy improvement, and thus operate under fundamentally different principles compared to SPI-based approaches. While our focus is on algorithms that offer theoretical guarantees under baseline constraints, we acknowledge the relevance of recent model-based offline RL methods, most notably MOReL (Kidambi et al. 2020), which adopt pessimism-driven strategies to avoid unsafe generalization. We explicitly consider MOReL as a point of comparison later in the paper.

3.1 MCTS-SPIBB

MCTS-SPIBB is a Monte Carlo Tree Search extension of SPIBB (Laroche et al. 2019). As MCTS approximates optimal policies generated by policy iteration, so MCTS-SPIBB approximates Π_0-optimal policies generated by SPIBB starting from a baseline π_0. The MCTS-SPIBB policy with infinite simulations is Π_0-optimal in the Maximum Likelihood Estimate (MLE) transition model $T^{\mathcal{D}}$ (Theorem 1) (Laroche et al. 2019) and a ζ-approximate safe policy improvement over π_0 (Theorem 2) (Laroche et al. 2019). Since MCTS-SPIBB computes the policy online and locally, it can scale to larger problems than SPIBB. The convergence of MCTS-SPIBB to the optimal policy in Π_0 w.r.t. $M^{\mathcal{D}}$ and the capability of MCTS-SPIBB to scale better than SPIBB are two key contributions shown in Section 3.1.3 (theoretical analysis) and Section 4 (experimental evaluation), respectively.

The main idea behind MCTS-SPIBB, explained in the following, is to extend UCT considering the constraint on bootstrapped actions. This is non-trivial for several reasons, e.g., UCT selects actions according to Q-values the constraint is on action selection probabilities, and the effect of the constraint accumulates in the layers of the MC tree. Bootstrapped actions must be selected with probability $\pi_0(s, a)$ during the simulations to generate optimal policies in Π_0. Figure 1 shows a diagram that highlights the key idea behind this extension. Given a state s in the MC tree, we split the actions into bootstrapped state-action pairs $(s, a) \in \mathcal{B}$ and non-bootstrapped state-action pairs, $(s, a) \in \overline{\mathcal{B}}$ (i.e., respectively, actions a_1 and a_2, and actions a_3 and a_4 in Figure 1). When the simulation reaches state s, we select a bootstrapped action with probability $p^s_{\mathcal{B}} = \sum_{a \in \mathcal{B}_A(s)} \pi_0(s, a)$, where $\mathcal{B}_A(s)$ is the set of bootstrapped actions for state s, and a non-bootstrapped action with probability $p^s_{\overline{\mathcal{B}}} = 1 - p^s_{\mathcal{B}}$. In the first case, we choose the specific action according to the probability distribution $\pi_0(s, \cdot)$. In the second case, we choose the specific action according to the UCT strategy, which considers the current estimates of Q-values and visit counts (respectively, $Q_3(s, a)$, $Q_4(s, a)$ and $N_3(s, a)$, $N_4(s, a)$ in Figure 1) and guarantees to select the optimal action with enough simulations. In rollout, baseline probabilities are used for bootstrapped actions and uniform selection for non-bootstrapped actions. At the end of the simulations, the estimated Q-values $Q(s, a)$ of the root state s are used to compute the probabilities of the improved policy $\pi^\circ(s,a)$ as *i)* $\pi_0(s,a)$ if $a \in \mathcal{B}(s)$, *ii)* $1 - p^s_{\mathcal{B}}$ if $a = \operatorname{argmax}_{a' \in \overline{\mathcal{B}}_A(s)} Q(s,a')$, *iii)* 0 otherwise.

3.1.1 Algorithm. Algorithm 4-7 show the pseudocode of MCTS-SPIBB. The differences with respect to standard MCTS are highlighted in blue. The agent is in a state s, and a baseline policy π_0 is available together with a dataset of trajectories $\mathcal{D}$ generated using π_0 on the real environment M^*. From dataset $\mathcal{D}$, we assume to have computed the state-action pair counts matrix $N_{\mathcal{D}}(s,a)$ and the MLE transition model $T^{\mathcal{D}}$. Other mathematical symbols present in the algorithms and already defined above are R, γ, $N_\wedge$, m, and the threshold ϵ used to end simulation steps. For the sake of compactness, these elements are used by the algorithms, although not as explicit input parameters.

Algorithm 4 MCTS-SPIBB

```
Input: s: current state; $\pi_0$: baseline policy; $N_\wedge$: minimum count; $N_{\mathcal{D}}$: counter; m: total number of simulations;
    $\mathcal{B}_A(s)$, $\overline{\mathcal{B}}_A(s)$: bootstrapped/non-bootstrapped actions
1:  Tr ← {} // Empty MC tree
2:  // Build MC tree (i.e., compute $Q(s,a)$)
3:  for $i = 1, \cdots, m$ do
4:      Simulate(Tr, $s$, 0, $\pi_0$, $\mathcal{B}_A(s)$, $\overline{\mathcal{B}}_A(s)$)
5:  end for
6:  $\pi^\circ(s,\cdot) \leftarrow (0,\cdots,0)$ // Initialize MCTS-SPIBB policy
7:  for $a \in \mathcal{B}_A(s)$ do
8:      $\pi^\circ(s,a) \leftarrow \pi_0(s,a)$
9:  end for
10: $a^\star \leftarrow \operatorname{argmax}_{a \in \overline{\mathcal{B}}_A(s)}$ {Tr.Q(s,a)}
11: $p^s_{\mathcal{B}} \leftarrow \sum_{a \in \mathcal{B}_A(s)} \pi_0(s,a)$ // $\mathcal{B}_A(s)$ total probability
12: $\pi^\circ(s,a^\star) \leftarrow 1 - p^s_{\mathcal{B}}$
13: return $a \sim \pi^\circ(s,\cdot)$
```

MCTS-SPIBB (**Algorithm 4**) first generates the Monte Carlo tree *Tr* for state s performing m simulations (lines 3-5). Then it produces the improved policy $\pi^\circ(s,a)$ setting *i)* the probabilities of bootstrapped actions to the related baseline probabilities (lines 7-9), *ii)* the probability of the best non-bootstrapped action $a^\star$ to the total probability available for non-bootstrapped actions (lines 10-12), *iii)* the probability of other non-bootstrapped actions to zero (line 6). Finally, it randomly samples an action from $\pi^\circ(s,\cdot)$ and returns it. Simulations (**Algorithm 5**) are performed using almost a standard MCTS strategy. Steps are performed using the MLE transition model T^D, and the simulator is set up as a standard MCTS simulator. **Algorithm 6** selects actions according to the *strategy described in Section 3.1.* It first decides whether to bootstrap or not, considering the total probability of bootstrapped actions $p^s_{\mathcal{B}}$ (lines 1-2). If it decides to bootstrap, it samples the action according to the probability distribution of those actions (lines 3-7). Otherwise, it samples the action according to standard UCT if the step is performed inside the tree (lines 9-11), or according to the rollout policy (we used a uniform policy in our tests) if the step is performed outside the tree (line 13). Finally, **Algorithm 7** performs the standard MCTS rollout using the new function for action selection (i.e., Algorithm 6).

What differentiates MCTS-SPIBB from the standard MCTS algorithm is how actions are selected both inside the tree and during rollouts. There are two non-trivial parts in the integration of SPIBB with MCTS. The first concerns the design of the action selection strategy (Figure 1); the second is the theoretical proof that this strategy, which merges UCT with baseline policy, actually provides a safe improvement (see next section and Appendix A).

Algorithm 5 Simulate

```
Input: Tr: MC tree structure; $s$: state node; $d$: current depth; $\pi_0$: baseline policy; $\mathcal{B}_A(s), \overline{\mathcal{B}}_A(s)$: bootstrapped/non-bootstrapped actions
1:  if $\gamma^d < \varepsilon$ then
2:      return 0
3:  end if
4:  // Node expansion
5:  if $s \notin$ Nodes then
6:      for $a \in \mathcal{A}$ do
7:          Nodes($sa$) $\leftarrow (N_{\text{init}}(s,a), Q_{\text{init}}(s,a), \emptyset)$
8:      end for
9:      return Rollout($s, d, \pi_0, \mathcal{B}_A(s), p^s_{\mathcal{B}}$)
10: end if
11: $p^s_{\mathcal{B}} \leftarrow \sum_{a \in \mathcal{B}_A(s)} \pi_0(s,a)$ // Tot prob bootstrapped act
12: $a^\star \leftarrow$ SelectAction($s, \mathcal{B}_A(s), \overline{\mathcal{B}}_A(s), \pi_0, p^s_{\mathcal{B}}, False$)
13: $s' \sim T^{\mathcal{D}}(s, a^\star, \cdot)$;  $r \leftarrow R(s, a^\star)$
14: $R \leftarrow r + \gamma \cdot$ Simulate(Tr, $s', d+1, \pi_0, \mathcal{B}_A(s'), \overline{\mathcal{B}}_A(s')$)
15: $N(s) \leftarrow N(s) + 1$
16: $N(s,a^\star) \leftarrow N(s,a^\star) + 1$
17: $Q(s,a^\star) \leftarrow Q(s,a^\star) + \frac{(R - Q(s,a^\star))}{N(s,a^\star)}$
18: return $R$
```

Algorithm 6 SelectAction

```
Input: $s$: state node; $\mathcal{B}_A(s), \overline{\mathcal{B}}_A(s)$: bootstrapped/non-bootstrapped action sets; $\pi_0$: baseline policy; $p^s_{\mathcal{B}}$: total probability of bootstrapped actions; $roll$: rollout flag
1:  $\theta \sim \mathcal{U}([0,1])$ // Uniform sampling from $[0,1]$
2:  if $\theta \leq p^s_{\mathcal{B}}$ then
3:      $p(\cdot) \leftarrow (0, \cdots, 0)$ // Init. bootstrapped probabilities
4:      for $a \in \mathcal{B}_A(s)$ do
5:          $p(a) \leftarrow \pi_0(s,a)/p^s_{\mathcal{B}}$
6:      end for
7:      $a^\star \sim p(\cdot)$ // Sample bootstrapped action
8:  else
9:      if $\neg roll$ then
10:         // Sample non-bootstrapped action using UCT
11:         $a^\star \leftarrow \text{argmax}_{a \in \overline{\mathcal{B}}_A(s)} \{Q(s,a) + 2C_p \sqrt{\frac{\log N(s)}{N(s,a)}}\}$
12:     else
13:         $a^\star \sim \pi_{rollout}(s, \cdot)$ // Sample uniformly
14:     end if
15: end if
16: return $a^\star$
```

Algorithm 7 Rollout

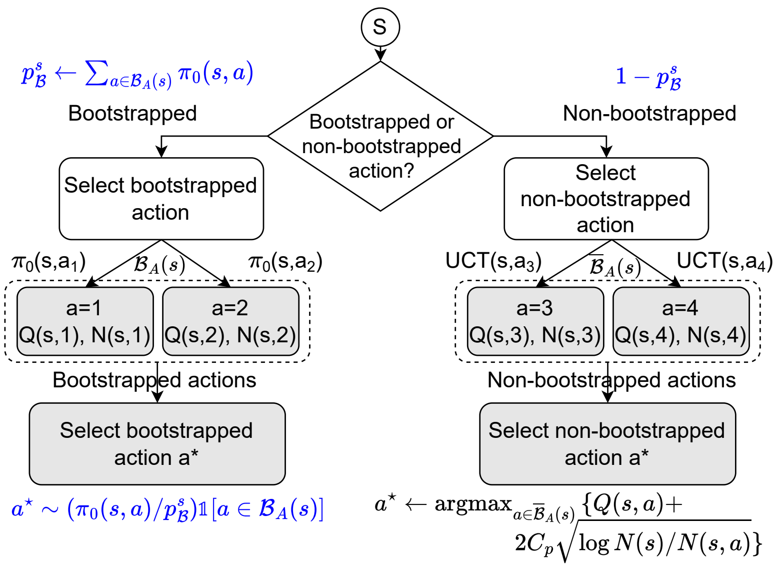
```
Input: $s$ : state node; $d$: current depth; $\pi_0$: baseline policy; $\mathcal{B}_A(s)$: bootstrapped action set; $p^s_{\mathcal{B}}$: total probability
    of bootstrapped actions
    if $\gamma^d < \varepsilon$ then
        return 0
    end if
    $a^\star \leftarrow$ SelectAction$(s, \mathcal{B}_A(s), \{\}, \pi_0, p^s_{\mathcal{B}}, True)$
    $s' \sim T^{\mathcal{D}}(s, a^\star, \cdot)$;  $r \leftarrow R(s, a^\star)$
    return $r + \gamma \cdot$ Rollout$(s', d+1, \pi_0, \mathcal{B}_A(s), p^s_{\mathcal{B}})$
```

Given a state s, the algorithm decides whether to select a bootstrapped or a non-bootstrapped action, and then it uses baseline probabilities (bootstrapped case) or Q-value estimates (non-bootstrapped case) to select actions. MCTS-SPIBB has convergence guarantees to SPIBB, and this would not have been ensured by other strategies.

A further important design choice concerns how the probability mass is distributed among the non-bootstrapped actions. In line with the original SPIBB formulation, MCTS-SPIBB assigns the entire probability mass to the single non-bootstrapped action having the upper bound of the Q-value estimated from simulations. During tree traversal, action selection is driven by the standard UCT criterion, i.e., by adding an exploration bonus term (Algorithm 6, line 6). Importantly, in each simulation, the method continues to sample bootstrapped actions according to the baseline policy's probability distribution for bootstrapped actions. This ensures that the improved policy remains within the constrained set Π_0 defined by SPIBB, and that it maximizes return over admissible policies. Alternative strategies, such as applying a softmax distribution over the Q-values of non-bootstrapped actions, could be of interest, but they would violate the SPIBB constraints and compromise the theoretical guarantee of safe improvement. Our approach, instead, preserves this guarantee and ensures convergence to the SPIBB-optimal policy. Theoretical justification is provided in Appendix A.

3.1.2 Complexity. The complexity of MCTS-SPIBB scales linearly with the number of simulations, namely is $O(m)$. This means that the computational effort required by MCTS-SPIBB is directly proportional to the number of simulations performed. Each simulation in MCTS-SPIBB is used to estimate the value of different actions by exploring potential future states. As the number of simulations increases, the accuracy of the action value estimates improves, but this also results in a corresponding linear increase in the computational complexity.

3.1.3 Theoretical Analysis. We prove that, given a baseline π_0, the improved policy π° generated by MCTS-SPIBB converges, as the number of simulations tends to infinity, to the improved policy π^{spibb} generated by SPIBB, which is optimal in Π_0 and a safe improvement of π_0. Using the notation of (Kocsis, Szepesvari, et al. 2006) and (Auer et al. 2002), which is derived from multi-armed bandit theory, we indicate with $X_{i,t}$ the (random) payoff (i.e., return) obtained by selecting action $i \in A$ in the t-th simulation passing from the current state. The average payoff of action i after m simulations passing from the current state is indicated as $\bar{X}_{i,m} \coloneqq \frac{1}{m}\sum_{t=1}^{m} X_{i,t}$ and the expected average payoff (i.e., the expected average return) of the current state after m simulations passing through it is indicated as $\mathbb{E}\{\bar{X}_m\}$. Among m simulations, n are assumed to select a non-bootstrapped action and c a bootstrapped action. $T_i(m)$ denotes the number of times action i was selected after m simulations (notation in Appendix A.1.1). The analysis is based on an assumption and three theorems. For the sake of compactness, the full assumption is reported in **Assumption A.5** (see Appendix A) and summarized here: without loss of generality, the expected value of the average payoff of each action $i \in A$ converges to some value $\mu_i \in \mathbb{R}$; payoffs $X_{i,t}$ are limited to range $[0, 1]$; the probability distributions over average payoffs concentrate quickly around their means, according to the Hoeffding inequality.



S
$p^s_{\mathcal{B}} \leftarrow \sum_{a\in\mathcal{B}_A(s)} \pi_0(s,a)$
$1 - p^s_{\mathcal{B}}$
Bootstrapped
Non-bootstrapped
Bootstrapped or non-bootstrapped action?
Select bootstrapped action
Select non-bootstrapped action
π_0(s,a$_1$)
$\mathcal{B}_A(s)$
π_0(s,a$_2$)
UCT(s,a$_3$)
$\overline{\mathcal{B}}_A(s)$
UCT(s,a$_4$)
a=1 Q(s,1), N(s,1)
a=2 Q(s,2), N(s,2)
a=3 Q(s,3), N(s,3)
a=4 Q(s,4), N(s,4)
Bootstrapped actions
Non-bootstrapped actions
Select bootstrapped action a*
Select non-bootstrapped action a*
$a^\star \sim (\pi_0(s,a)/p^s_{\mathcal{B}})\mathbb{1}[a \in \mathcal{B}_A(s)]$
$a^\star \leftarrow \text{argmax}_{a\in\overline{\mathcal{B}}_A(s)}\{Q(s,a)+ 2C_p\sqrt{\log N(s)/N(s,a)}\}$

Fig. 1. Action selection strategy of MCTS-SPIBB: flow chart.

The first theorem provides a bound on the bias of the estimated value (i.e., expected average return) for the current state after m simulations $\mathbb{E}\{\bar{X}_m\}$, given the bias on the estimated value of the optimal non-bootstrapped action $\delta^*_n \coloneqq \mathbb{E}\{\bar{X}_{i^*,n}\} - \mu^*$ (with μ^* expected value of the average payoff of the optimal non-bootstrapped action) and the biases on the estimated values of the bootstrapped actions $\delta_{i,T_i(c)} \coloneqq \mathbb{E}\{\bar{X}_{i,T_i(c)}\} - \mu_i$ (with $i \in \mathcal{B}_A(s)$, μ_i expected value of the average payoff of bootstrapped action i). $N_0(\epsilon)$ is such that if $t \geq N_0(\epsilon)$ then $|\delta_{i,t}| \leq \epsilon\Delta_i/2$ and $|\delta_{j^*,t}| \leq \epsilon\Delta_i/2$, where $\Delta_i = \mu^* - \mu_i$, with i sub-optimal action and j^* optimal action.

Theorem 3.1. *Let* $\bar{X}_m = \frac{1}{m}\sum_{i\in\mathcal{B}_A} T_i(c)\bar{X}_{i,T_i(c)} + \frac{1}{m}\sum_{i\in\overline{\mathcal{B}}_A} T_i(n)\bar{X}_{i,T_i(n)}$. *Under Assumption A.5 the following bound holds* $\left|\mathbb{E}\{\bar{X}_m\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot\mu_i - p_{\overline{\mathcal{B}}}\cdot\mu^\star\right| \leq \sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}| + p_{\overline{\mathcal{B}}}\cdot|\delta^*_n| + O\left(\frac{K(C_p^2\ln m + N_0)}{m}\right)$ *where* $N_0 = N_0(\epsilon)$.

Proof. *(Sketch)* The idea behind the proof is to split the bias on the expected payoff of the state (left side of the inequality) into the bias of the expected payoff of the optimal non-bootstrapped action and the bias of each bootstrapped action. The first is bounded by Theorem 3 of (Kocsis, Szepesvari, et al. 2006) and the second by $\sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}|$. Theorem A.6 (see Appendix A) provides the full derivation. □

The second theorem proves the convergence of the estimated state value $\mathbb{E}\{\bar{X}_m\}$ to the optimal value in Π_0, which is the value of the optimal policy in Π_0 computed by SPIBB.

Theorem 3.2. *Consider algorithm MCTS-SPIBB running on a tree of depth D, branching factor $|A| = L + K$ with L bootstrapped actions and K non-bootstrapped actions in each state, and stochastic payoffs at the leaves. Assume that payoffs lie in* $[0,1]$. *Then the bias of the estimated expected payoff $\bar{X}_m$ is* $O\left(\frac{L^D KD\ln m + L^D K^D}{m}\right)$.

Proof. *(Sketch)* The proof is made by induction on D considering the bound of Theorem 3.1 to perform the inductive step from $L+K$ MC trees of depth $D-1$ (one tree for each non-bootstrapped action and one tree for each bootstrapped action) to a MC tree of depth D. Theorem A.7 (see Appendix A) provides the full derivation. □

The third theorem proves that the estimated state value concentrates quickly around its mean.

Theorem 3.3. *Fix an arbitrary $\delta > 0$ and let $\Delta_m = 9\sqrt{2p^{s}_{\overline{\mathcal{B}}} m \ln(4/\delta)} + C_p\sqrt{m\ln(2L/\delta)} - \sum_{i\in\mathcal{B}_A}\sqrt{\pi_0(i)}$. Let $n_0 \in \mathbb{N}$ be such that $\sqrt{n_0} \geq O(K(C_p^2 \ln n_0 + N_0(1/2)))$, if $m \geq \frac{n_0}{p_{\overline{\mathcal{B}}}}$ then under Under Assumption A.5 the following bounds hold: $\mathbb{P}\left\{m\bar{X}_m \geq m\mathbb{E}\left\{\bar{X}_m\right\} + \Delta_m\right\} \leq \delta$ and $\mathbb{P}\left\{m\bar{X}_m \leq m\mathbb{E}\left\{\bar{X}_m\right\} - \Delta_m\right\} \leq \delta$.*

Proof. *(sketch)* The proof is obtained by splitting each probability $\mathbb{P}\{\cdot\}$ into two terms, one for non-bootstrapped actions and one for bootstrapped actions. Bounds on the probability distribution of average payoffs in Assumption A.5 and the Hoeffding inequality allow to prove the theorem. Details are reported in Theorem A.8 (see Appendix A). □

This theoretical analysis shows the safety of MCTS-SPIBB by demonstrating the convergence of the policy generated by MCTS-SPIBB to the policy generated by SPIBB (which is proved to be safe in (Laroche et al. 2019)). Furthermore, SPIBB computes an optimal policy in Π_0 (Theorem 1 (Laroche et al. 2019)), namely, it solves optimally the problem of SPI with baseline bootstrapping (satisfying the percentile criterion of Eq. 1 (Laroche et al. 2019)).

3.2 SDP-SPIBB

We start observing that the policy evaluation step performed by DP-SPIBB (see Algorithm 3) is inefficient. In fact, in standard policy iteration, the four loops over states, actions, next states, and next actions, require $|S|^2|A|^2$ updates of the ψ term (see line 10 in Algorithm 3). However, differently from the general policy iteration context, in the SPI context, the non-null updates of this term are only those for which $T^{\mathcal{D}}(s, a, s')$ is different from 0, namely, the updates for which triplets (s, a, s') are present in dataset $\mathcal{D}$. For this reason, the complexity of policy evaluation in the SPI context can be reduced from $|S|^2|A|^2$ to a term depending on the dataset size $|\mathcal{D}|$, and specifically on the non-bootstrapped state-action set $\overline{\mathcal{B}}$, which is derived from the dataset and represents pairs of states and actions that have been observed at least $N_\wedge$ times. This approach allows for more efficient policy improvement by concentrating on the most relevant data, thus optimizing computational resources. In particular, we modify the Q-table update procedure of Algorithm 3 to consider only terms (i.e., triplets (s, a, s')) present in $\mathcal{D}$, which have $T^{\mathcal{D}}(s, a, s') \neq 0$. The number of these terms is usually much smaller than $|S|^2|A|^2$ in large domains since the number of possible transitions is huge, but the agent can realistically explore a small number of them.

3.2.1 *Algorithm.*

The formalization of SDP-SPIBB is provided in **Algorithm 8**. Both input and output remain consistent with DP-SPIBB. Initially, it defines the set $SA^{\overline{\mathcal{B}}}$, as the set of state-action pairs (s, a) present in the non-bootstrapped set $\overline{\mathcal{B}}$. SDP-SPIBB restricts its exploration to this specific set of pairs $(s, a) \in SA^{\overline{\mathcal{B}}}$ (Algorithm 8, line 6), thereby avoiding exploring the entire state and action spaces. In this context, the return from the next state ψ is initialized to zero. Furthermore, the set $NSA^{\overline{\mathcal{B}}}$ is defined, containing state-action pairs (s', a') such that $(s, a, s', \cdot) \in \overline{\mathcal{B}}$ and $(s', a', \cdot, \cdot) \in \overline{\mathcal{B}}$ (Algorithm 8, line 8). A loop is initiated over the next reachable state-action pairs $(s', a') \in NSA^{\overline{\mathcal{B}}}$, with rewards accumulated for each state (Algorithm 8, lines 10-12). The Q-table is updated as in DP-SPIBB, but considering only a subset of cases.

The policy improvement is performed through a novel function (see **Algorithm 9**), which differs from the one in Algorithm 2 in terms of the set of state-action pairs it iterates over. Specifically, the policy improvement procedure of DP-SPIBB iterates over all states (Algorithm 3, line 14), whereas the policy improvement procedure of SDP-SPIBB (Algorithm 9) consistently iterates only on the state-action pairs in $SA^{\overline{\mathcal{B}}}$ (Algorithm 9, line 2). Consequently, SDP-SPIBB seeks improvement for the (s, a) pairs in the Q-table only where updates can be made. The proposed modification guarantees the equivalence between SDP-SPIBB and DP-SPIBB since SDP-SPIBB only avoids iterating over triplets (s, a, s') that do not influence the update of the Q-table.

Algorithm 8 SDP-SPIBB

Input: baseline policy π_0, transition matrix $T^{\mathcal{D}}$, discount factor γ, stopping parameter ϵ
Output: an improved policy π

```
1: $SA^{\overline{\mathcal{B}}} = \{(s,a)|(s,a,\cdot,\cdot) \in \overline{\mathcal{B}}\}$ // set $(s,a) \in \overline{\mathcal{B}}$
2: $\pi = \pi_0$ // Initialization to the baseline policy
3: $Q = 0$
4: $\delta_Q = \infty$
5: while $\delta_Q > \epsilon$ do
6:     for $(s,a)$ in $SA^{\overline{\mathcal{B}}}$ do
7:         Initialize $\psi = 0$ // Return from the next states
8:         $NSA^{\overline{\mathcal{B}}} = \{(s',a') \mid (s,a,s',\cdot) \in \overline{\mathcal{B}} \wedge$
               $(s',a',\cdot,\cdot) \in \overline{\mathcal{B}}\}$
9:         for $s',a'$ in $NSA^{\overline{\mathcal{B}}}$ do
10:            $\psi = \psi + Q(s',a')\pi(s',a')T^{\mathcal{D}}(s,a,s')$
11:        end for
12:        $Q'(s,a) = R(s,a) + \gamma\psi$
13:    end for
14:    $\pi$ = Scalable Policy Improvement$(Q', \pi_0, SA^{\overline{\mathcal{B}}})$
15:    $\delta_Q = \|Q - Q'\|_\infty$
16:    $Q = Q'$
17: end while
```

Algorithm 9 Scalable Policy Improvement on $SA^{\overline{\mathcal{B}}}$

Input: last iteration of Q, baseline policy π_0, $SA^{\overline{\mathcal{B}}}$
Output: policy improvement π^I

```
1: $\pi^I = \pi_0$ // Initialization to the baseline policy
2: for $(s,a)$ in $SA^{\overline{\mathcal{B}}}$ do
3:     $\pi^I(s,a)_{a|(s,a)\in\mathcal{B}} = \pi_0(s,a)_{a|(s,a)\in\mathcal{B}}$
4:     $\pi^I(s, \operatorname{argmax}_{a|(s,a)\in\overline{\mathcal{B}}} Q(s,a)) = \sum_{a|(s,a)\in\overline{\mathcal{B}}} \pi_0(s,a)$
5: end for
```

3.2.2 Complexity. The complexity of SDP-SPIBB depends on the cardinality of the non-bootstrapped set $\bar{\mathcal{B}}$, and is formally $O(|\bar{\mathcal{B}}|)$ (see Algorithm 8, lines 6–13, and Algorithm 9, lines 2–5). While in the worst-case scenario, where nearly all state-action-next-state triplets are sufficiently supported, this may approach the complexity of standard DP-SPIBB, namely $O(|S|^2|A|^2)$, such cases are rare in practice. In large and structured domains, datasets are typically sparse and concentrated on a relatively small subset of transitions, and the behavior policy itself often fails to provide adequate coverage of the full state-action space. As a result, $|\bar{\mathcal{B}}|$ tends to be several orders of magnitude smaller than the full state-action-next-state space. This practical sparsity becomes even more evident in high-dimensional problems, where full state-action coverage is rarely achieved, and safe updates remain limited to well-supported regions. However, since the complexity still depends on the dataset size, it can grow in applications where huge amounts of data are collected over time (e.g., streaming data).

3.2.3 Theoretical Analysis. Here we show that SDP-SPIBB is equivalent to SPIBB. The key observation of this equivalence is that the policy evaluation and policy improvement functions of DP-SPIBB (Algorithm 3 and 2) and SDP-SPIBB (Algorithm 8 and 9) perform the same operations on the Q-table, only SDP-SPIBB avoids performing operations that do not affect the Q-table because they are not supported by observations in the trajectory dataset. Hence, the policies generated by SPIBB and SDP-SPIBB are equal. The update of the Q-table referred to as Q' in the algorithms (lines 12 and 13, respectively), are only influenced by state-action pairs present in the dataset $\mathcal{D}$. The equivalence between the algorithms is formalized by the following theorem.

Theorem 3.4. *Let π_0 and $M^{\mathcal{D}}$ be a baseline policy and a MLE model generated from dataset $\mathcal{D}$. If π and π' are the policies improved generated by DP-SPIBB (Algorithm 3) and SDP-SPIBB (Algorithm 8) respectively, then $\pi(s,a) = \pi'(s,a)$, $\forall s \in S$ and $\forall a \in A$.*

Sketch of proof. A close examination of Algorithm 3 shows that during the loops in lines $5-6$, which iterate over all states and actions in the domain, two scenarios emerge: i) if state-action pairs $(s,a) \in \mathcal{D}$, then the Q-table Q' update for (s,a) relies on the sum of the immediate reward caused by action a in state s and the return from the next states (Algorithm 3, line 11); ii) if $(s,a) \notin \mathcal{D}$, or equivalently, if the probability distribution of the next states $T^{\mathcal{D}}(s,a,\cdot)$ is unknown, the algorithm assumes no available transition. In other words, for each $s' \in S$, where s' is considered a terminal state, the return from the next states $\psi = 0$ (Algorithm 3, line 9). In this latter case, the Q-table update is determined only by the immediate reward provided by the reward function, given that the number of known transitions is restricted and depends on the size of $\mathcal{D}$.

In the policy improvement phase, SPIBB allows for the improvement of the baseline policy and consequently computes a policy improvement only over the set of state-action pairs belonging to $\overline{\mathcal{B}}$ (i.e., the non-bootstrapping state-action pairs). If the probability distribution of the next states for state s is unknown, this can be attributed to one of three factors: s not being observed in $\mathcal{D}$, the baseline reaching a terminal state, or the baseline terminating its execution in that state. In such cases, the entire set of actions in state s belongs to the set of bootstrapped actions $\mathcal{B}$. Consequently, any execution leading to state s would involve the use of the baseline policy. Additionally, SPIBB cannot modify the Q-table for state s, and thus, no opportunities for policy improvement would exist. On the other side, DP-SPIBB explores the complete state and action space, attempting to update the Q-table even for pairs (s,a) not present in $\mathcal{D}$. As a result, a significant number of Q-table update attempts would be executed unnecessarily by DP-SPIBB, which leads to higher computational costs and possibly missed convergence in large domains. In Algorithm 3, when Q is multiplied by the transition model T (line 10), the state–action pairs not represented in the MLE model satisfy $T^{\mathcal{D}}(s,a,s') = 0$. This implies that updates outside the non-bootstrapped set have no effect on the resulting policy in the policy improvement step. Consequently, although SDP-SPIBB explicitly restricts updates to $\bar{\mathcal{B}}$, the outcome is equivalent to DP-SPIBB, which formally includes all state–action pairs, since the terms corresponding to the bootstrapped set cancel out.

4 Experimental Evaluation

In this section, we first report an overview of the experiments performed to evaluate MCTS-SPIBB and SDP-SPIBB. Then, we provide details on the benchmark domains, the procedure used to generate baseline policies, the overall experimental procedure and the baseline algorithms to which our methods have been compared. Finally, we analyze the results on the scalability and safety of the two proposed methods, showing that they scale to larger domains than baseline methods while keeping safety guarantees. An ablation study dissects the sensitivity of the proposed methods to key experimental factors, such as dataset diversity and the number of simulations. All tests can be reproduced by the Python code available in the repository connected to this paper[2]

[2]https://github.com/Isla-lab/scalable_SPI

4.1 Experiment Overview

We conducted a series of experiments to evaluate the performance of MCTS-SPIBB and SDP-SPIBB across several domains, focusing on their scalability and safety. Our goal is to demonstrate empirically that MCTS-SPIBB and SDP-SPIBB scale efficiently to larger domains than standard SPIBB and state-of-the-art SPI algorithms while maintaining safety guarantees.

Scalability. The evaluation of the scalability of the algorithms (Section 4.6) is conducted on both single-agent and multi-agent domains as described below.

- *Single-agent domains:* In the single-agent case, we evaluate the scalability of the algorithms on SysAdmin as the domain size varies, using the benchmark proposed in (Castellini et al. 2023). In this domain, the state space size grows exponentially with the number of machines. We compare our methods with SPIBB (I-SPIBB) and DP-SPIBB, given a baseline policy. The results show that MCTS-SPIBB and SDP-SPIBB can scale to large domain sizes to which state-of-the-art techniques cannot be applied for complexity reasons.
- *Multi-agent domains:* In the multi-agent case, we evaluate the scalability of the algorithms on the multi-agent SysAdmin domain as the number of agents increases, using the benchmark proposed in (Choudhury et al. 2021). In this domain, both the state space and the action space sizes grow exponentially with the number of agents. The comparison is only possible between MCTS-SPIBB and SDP-SPIBB, as SPIBB, DP-SPIBB, and state-of-the-art SPI methods based on policy iteration are not able to scale even to the smallest number of agents considered in the experiments. The results show that both MCTS-SPIBB and SDP-SPIBB can scale but MCTS-SPIBB maintains a smaller computational time than SDP-SPIBB as the number of agents increases. This is a key result showing the superiority of MCTS-SPIBB on very large domains.

Safety. The evaluation of the safety of our methods (Section 4.7) is conducted first with all state-of-the-art SPI and with some of the leading model-based/model-free offline RL methods. Subsequently, the comparison focuses specifically on SPIBB, as MCTS-SPIBB and SDP-SPIBB are derived from this algorithm.

- *Comparison with state-of-the-art SPI methods:* In this experiment, we compare the safety of MCTS-SPIBB and SDP-SPIBB with that of all state-of-the-art SPI algorithms in the benchmark presented by (Scholl et al. 2022a), which is performed on the Wet-Chicken domain. The results show that both MCTS-SPIBB and SDP-SPIBB ensure safety and, in particular, achieve state-of-the-art performance with respect to safety guarantees.
- *Comparison with model-free/model-based offline RL methods:* In this experiment, we compare MCTS-SPIBB and SDP-SPIBB with state-of-the-art methods spanning model-free, model-based, and offline RL, namely PPO, DQN, CQL, Decision Transformer, and MOReL. Although some of these methods differ fundamentally in their principles and are not designed to provide PAC-style safety guarantees, they offer valuable points of comparison to assess empirical performance and the potential for safety violations in practice. We conduct this evaluation on the benchmark presented by (Scholl et al. 2022a), as done in the comparison with all state-of-the-art SPI algorithms. The results show that MCTS-SPIBB and SDP-SPIBB maintain stable and reliable performance even under limited data conditions, unlike the additional methods considered, which show high variance and frequent performance degradation in low-data regimes due to the lack of safety constraints.
- *In-depth comparison with SPIBB:* In this experiment, we focus the comparison on safety on SPIBB using the benchmark presented by (Castellini et al. 2023; Laroche et al. 2019), which includes GridWorld and SysAdmin domains. This comparison is made specifically with SPIBB to evaluate the performance of MCTS-SPIBB and SDP-SPIBB relative to the method that uses the same safety criterion as our methods. The results show that both MCTS-SPIBB and SDP-SPIBB are safe. Furthermore, the performance of MCTS-SPIBB converges to that of SPIBB as the number of simulations increases, and the performance of SDP-SPIBB matches that of SPIBB if the same convergence conditions are considered.

Ablation Study. We perform an in-depth analysis of the sensitivity of MCTS-SPIBB and SDP-SPIBB performance (Section 4.8) under varying dataset and algorithmic conditions. All ablation studies are conducted on the SysAdmin domain with 7 machines, which provides a sufficiently complex environment for analyzing algorithmic behavior under controlled changes. Specifically, we examine the effect of the number of trajectories available in the offline dataset under a random baseline policy, the impact of baseline policy quality and dataset diversity on performance, and the robustness of MCTS-SPIBB to different simulation budgets.

- *Performance sensitivity to dataset size under a random baseline:* In this experiment, we investigate how the number of trajectories in the offline dataset affects the ability of MCTS-SPIBB, SDP-SPIBB and SPIBB to safely improve over a random baseline policy. The results show that, although performance is initially close to the baseline when data is scarce, all the algorithms achieve increasingly better policies as the dataset size grows, converging toward the optimal policy while maintaining safety guarantees even in low-data regimes.
- *Sensitivity to baseline policy and dataset coverage:* In this experiment, we investigate how the diversity of datasets generated by a baseline policy affects performance improvement. We consider two types of baseline policies: one that chooses the optimal action with a probability of $p = 0.5$ (which we call the random baseline) and one that chooses the optimal action with a probability of $p = 0.9$ (which we call the sub-optimal baseline). The results show that MCTS-SPIBB, SDP-SPIBB and SPIBB have equivalent performance in both cases.
- *Robustness of MCTS-SPIBB to simulation budget variation:* In this experiment, we specifically examine the number of simulations required for MCTS-SPIBB to achieve optimal performance starting from a random or sub-optimal baseline policy. The results show that MCTS-SPIBB achieves optimal performance in both cases; however, convergence to the improved optimal policy is faster when the algorithm uses the sub-optimal baseline.

4.2 Domains

In the following, all the domains used in the experimental evaluation are listed and described in detail.

GridWorld. In *GridWorld* (Laroche et al. 2019) an agent moves in a $N \times N$ grid starting from the bottom-left corner and aiming to reach the top-right corner. The agent can select four actions, i.e., moving north, south, east, or west. Each action has a 75% chance of moving the agent in the desired direction, 5% in the opposite direction, and 10% chance of moving it in each of the other two directions. The reward is 1 if the agent reaches the target cell (top-right corner), 0 otherwise. The size of the state space is $|S| = N^2$ and that of the action space is $|A| = 4$.

SysAdmin. In *SysAdmin* (Guestrin et al. 2003) an agent has to administer a network of N machines. Each machine is connected to two other machines on either side of it to form a ring topology. A binary random variable represents whether each machine is working or has failed. At each time step, the agent can reboot one machine for -1 or do nothing with null cost, furthermore, it receives a reward of 1 for each working machine and a penalty of -1 for each failed machine. Every machine has a probability of 0.05 to fail at each time step. This probability increases by 0.3 for each neighbouring machine that failed. If a machine is rebooted, then it works with probability 1. The size of the state space is $|S| = 2^N$, and the size of the action space $|A| = N + 1$.

WetChicken. In *WetChicken*(Scholl et al. 2022a) an agent floats in a small boat on a river with a waterfall at one end. The goal for the agent is to stay as close as possible to the waterfall without falling. The closer the agent is to the waterfall, the greater the reward. If it falls, the episode ends. The river is represented as a 5×5 grid in the benchmark (i.e., $|S| = 25$) and five actions can be chosen by the agent.

Multi-agent SysAdmin. In *Multi-agent SysAdmin* (Guestrin et al. 2003), each agent controls a single machine in a network of interconnected machines. Each machine is described by two variables: *status* (*good*, *faulty*, or *dead*) and *load* (*idle*, *loaded*, or *success*). Initially, all machines are *good* and *idle*. Agents can, at each time step, activate their machines, or take no action. The actions may result in status changes, and agents are rewarded for machines reaching the *(good, success)* state. The state and action spaces grow exponentially with the number of agents n, with state space size $|S| = 9^n$ and action space size $A = 2^n$.

4.3 Generation of the Baseline Policy

The generation of the baseline policy occurs in two ways, depending on the size of the considered domain. For small versions of GridWorld, SysAdmin, and Wet-Chicken, the baseline policy is generated by first computing an optimal policy π_{opt} with policy iteration, running it on the domain, and then applying random noise to the probabilities to introduce sub-optimal choices. For large SysAdmin and multi-agent SysAdmin domains, used to test scalability, the computation of optimal policies by policy iteration is not possible; hence, the baseline policies are generated from simple rules. For instance, in the SysAdmin domain, given the configuration of machines being turned on/off in a specific state, the rule states that: if all machines are off, choose randomly to turn on a machine with a probability of $p = \frac{1}{n}$, where n is the number of machines; if at least one machine is on, turn on the closest machine that is off with probability p or do nothing with probability $1 - p$; if the turned-on machine has two adjacent machines that are off, it randomly chooses one of the two; if all machines are on, do nothing. In multi-agent SysAdmin, each agent controls its machine and, if it is off, decides to turn it on with a probability of p or do nothing with a probability of $1 - p$. Full details about baseline policy generation can be found in the code repository connected to this paper.

4.4 Experimental Procedure

All the experiments begin by generating a baseline policy in the two ways explained in Section 4.3. After that, the baseline policy is run in the true environment M^* to collect data. Depending on the experiment, datasets of different sizes are collected. For each dataset size $|\mathcal{D}|$, we generate ND datasets, each containing $|\mathcal{D}|$ trajectories. Afterwards, for each dataset, we compute the estimated MLE transition model $T^{\mathcal{D}}$, the counters $N_{\mathcal{D}}(s, a)$, and sets of bootstrapped/non-bootstrapped state-actions $\mathcal{B}_A(s)/\overline{\mathcal{B}}_A(s)$ using the threshold $N_\wedge$. This threshold is not a property of the dataset but of the environment, as defined in SPIBB (Laroche et al. 2019). Its theoretical value depends on $V_{\max}$, $|S|$, $|A|$, and other MDP parameters, and represents an upper bound guaranteeing safe policy improvement. In practice, in our experiments and prior work adopt smaller values of $N_\wedge$ that still yield safe improvements, although no general rule exists to identify the minimal safe threshold.

The parameter ϵ of SDP-SPIBB corresponds to the standard stopping criterion in policy iteration, which controls the precision of Q-value convergence. As in standard policy iteration, the optimality of the policy is guaranteed only with $\epsilon = 0$, and the policy generated by policy iteration approximates the optimal policy with a precision level that depends on ϵ. Therefore, ϵ is not specific to SDP-SPIBB but a general hyper-parameter that is also present in SPIBB, and it can be tuned depending on the desired trade-off between computational cost and approximation accuracy. Our methods inherit the same safety guarantees of SPIBB when using the same ϵ. A full analysis of the dependence between ϵ and the safety guarantees of SPIBB is out of the scope of this paper. In our experiments, we set $\epsilon = 10^{-9}$, following the choice in SPIBB, to ensure convergence.

Finally, for each dataset, we generate the improved policy in the MLE MDP $M^{\mathcal{D}}$ using the SPI algorithms. Those algorithms are evaluated on the true environment M^* in two possible ways: i) on small domains, we compute a Policy Evaluation on the improved policy as $\rho(\pi_I, M^*) = V_{M^*}^{\pi_I}(s_0)$, where s_0 is the initial state; ii) on large domains, where Policy Evaluation cannot be used because of its complexity dependent on state and action spaces, we compute the average discounted return of the improved policy as $\bar{\rho}(\pi_I, M^*) = V_{M^*}^{\pi_I}(s_0)$, running the improved policy in M^* for a certain number of runs.

4.5 Reference Algorithms for Evaluating Safety and Scalability

To assess scalability in single-agent domains, we compare our proposed methods, MCTS-SPIBB and SDP-SPIBB, with SPIBB (Laroche et al. 2019) and DP-SPIBB (Laroche et al. 2019)[3], focusing the comparison on SPI methods that offer PAC-style safety guarantees. The goal is to evaluate the scalability of these approaches on large-scale domains. In multi-agent domains, only MCTS-SPIBB and SDP-SPIBB are evaluated, as they are the only methods among those considered that scale effectively to large multi-agent environments. To evaluate safety, we thoroughly benchmark MCTS-SPIBB and SDP-SPIBB against the state-of-the-art SPI algorithm SPIBB (Laroche et al. 2019). In addition, we compare the safety of our methods with all the leading SPI algorithms included in the benchmark proposed by (Scholl et al. 2022a)[4]. Specifically, we consider DUIPI (Schneegass, Hans, et al. 2010), R-MIN (Brafman and Tennenholtz 2003), RaMDP (Petrik et al. 2016), and several variants of SPIBB as discussed in (Scholl et al. 2022a). These algorithms do not provide conservative safe policy improvement guarantees like SPIBB, and are included to evaluate the trade-off between performance and safety.

For completeness, and in line with prior literature, we also include a baseline we refer to as Basic-RL, following the terminology used in (Laroche et al. 2019). Basic-RL is a standard Policy Iteration algorithm that uses a transition model estimated via Maximum Likelihood Estimation from the offline dataset, without applying any bootstrapping or safety constraints. Specifically, it is allowed to optimize freely over all state-action pairs, regardless of the level of data support. This baseline is included to show the potential risks associated with aggressive optimization over a possibly biased model. As additional baselines, we include DQN (Mnih et al. 2013) and PPO (Schulman et al. 2017)[5], two widely used model-free deep reinforcement learning algorithms. We also consider CQL (Kumar, Zhou, et al. 2020)[6], a state-of-the-art offline RL method, and the more recent Decision Transformer (Chen et al. 2021)[7]. For direct comparison with a state-of-the-art model-based offline RL method, we included MOReL (Kidambi et al. 2020)[8] in our experimental setup. Although some of these methods are originally online algorithms, their offline adaptation, using fixed datasets without environment interaction, has been previously studied in offline RL benchmarks(Fujimoto et al. 2019). These algorithms are trained exclusively on offline data using the MLE-estimated transition model obtained from the dataset as a simulator of the real environment, when required, except for MOReL. Since MOReL is a model-based offline RL method, it learns the dynamics through an ensemble of nets, and therefore, we only provided it with the data collected by executing the baseline policy in the real environment. Our motivation is twofold: i) to ensure fair comparisons with our SPIBB-based methods using the same data and environment models, and ii) to assess how standard model-free algorithms behave under purely offline constraints.

4.6 Scalability

In this section, we assess the scalability of MCTS-SPIBB and SDP-SPIBB in both single-agent and multi-agent domains.

4.6.1 Scalability on Single-Agent Domains. We evaluate the scalability of MCTS-SPIBB and SDP-SPIBB on the benchmark proposed in (Castellini et al. 2023), considering large-scale single-agent SysAdmin and varying the number of machines from 4 to 35. For each case, i.e., for each number of machines, we generated 20 datasets of 5000 trajectories using a baseline policy that selects the optimal action with a probability $p = 0.7$. The algorithms are evaluated in each case using the average discounted return $\bar{\rho}(\pi_I, M^*) = V_{M^*}^{\pi_I}(s_0)$, over 20 runs

[3]https://github.com/RomainLaroche/SPIBB

[4]https://github.com/Philipp238/Safe-Policy-Improvement-Approaches-on-Discrete-Markov-Decision-Processes

[5]https://stable-baselines.readthedocs.io/en/master/

[6]https://corl-team.github.io/CORL/

[7]https://huggingface.co/

[8]https://github.com/SwapnilPande/MOReL

of 15 steps each. For the algorithms, we also show the times required to compute the policy improvement. All parameters of this experiment are available in Table 2 in Section C.1 of the appendix. In Figure 2.a, the box

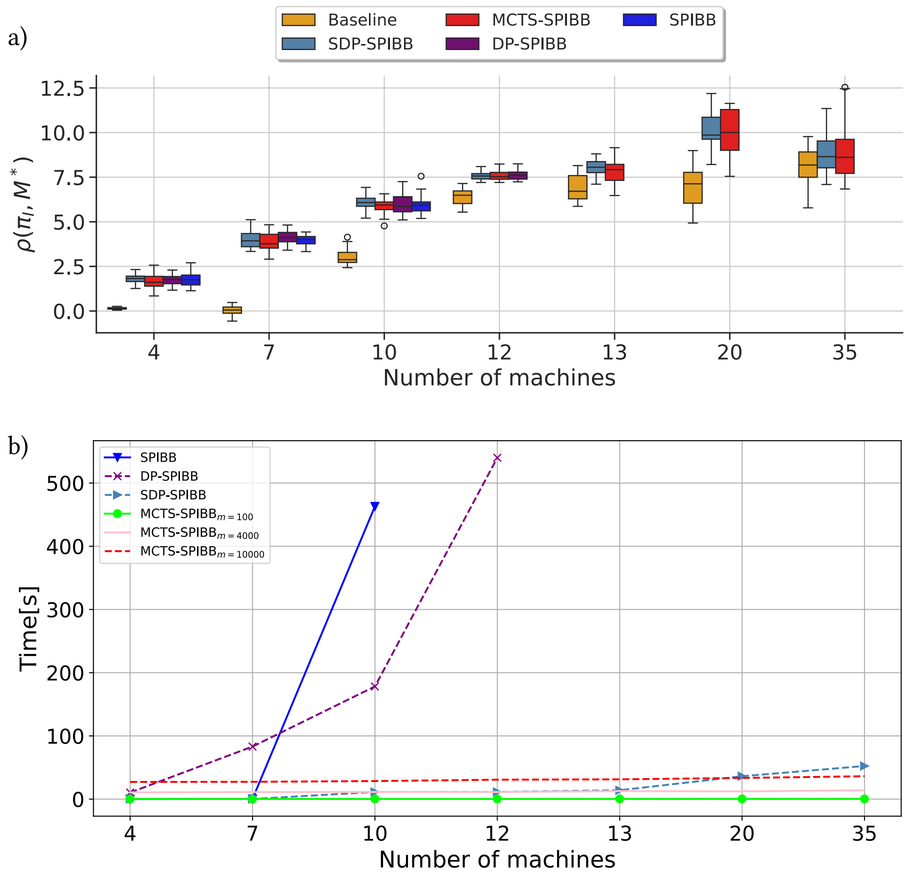
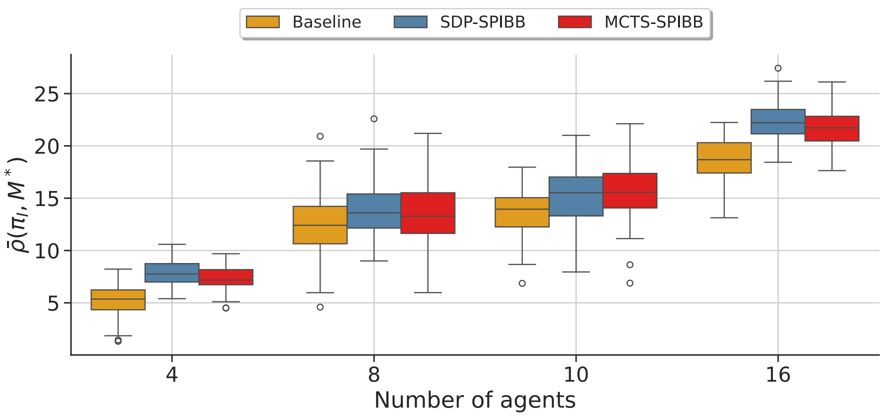


Fig. 2. Scalability on single-agent SysAdmin: a) performance (y-axis) of the baseline policy ($p = 0.7$, yellow boxplots), SDP-SPIBB (steel-blue boxplots), MCTS-SPIBB (red boxplots), DP-SPIBB (purple box-plots) and SPIBB (blue box-plots) as the number of machines varies (x-axis); b) average computational time (y-axis) of SDP-SPIBB, MCTS-SPIBB, DP-SPIBB and SPIBB as the number of machines varies (x-axis).

plots show the performance (y-axis) of the algorithms as the number of machines increases (x-axis). I-SPIBB generates improved policies only up to 10 machines, while DP-SPIBB scales up to 12 machines. In contrast, MCTS-SPIBB and SDP-SPIBB scale further, up to 35 machines. I-SPIBB has a complexity of $O(|S|^3 \times |A|^3)$ (it is solved by matrix inversion) and DP-SPIBB has a complexity $O(|S|^2|A|^2)$ (it uses dynamic programming) and this makes the algorithms based on policy iteration less scalable to large problems. In contrast, MCTS-SPIBB has a complexity of $O(m)$, with m number of simulations, and it does not directly depend on the state/action space size. SDP-SPIBB's complexity, even though the algorithm is also based on policy iteration, depends on the size of the non-bootstrapped set $\overline{\mathcal{B}}$, hence it is $O(\overline{\mathcal{B}})$. In Figure 2.b, the lines show the mean computational times (y-axis) as the number of machines varies (x-axis). SPIBB and DP-SPIBB exhibit exponential growth in computation time with the number of machines, whereas MCTS-SPIBB maintains a constant computational time that depends only on the number of simulations m. SDP-SPIBB scales better than SPIBB and DP-SPIBB, although it starts to show increased computation times with 20 and 35 machines.

4.6.2 Scalability on Multi-Agent Domains. We evaluate the scalability of MCTS-SPIBB and SDP-SPIBB on the benchmark proposed in (Choudhury et al. 2021), considering the multi-agent SysAdmin domain and varying the number of agents, i.e., 4, 8, 10, 16. For each case, i.e., for each number of agents, we generated 20 datasets of [10000, 50000, 100000, 200000, 500000] trajectories using a baseline policy that selects the optimal action with a probability $p = 0.7$. The algorithms are evaluated in each case using the average discounted return $\bar{\rho}(\pi_I, M^*) = V^{\pi_I}_{M^*}(s_0)$, over 20 runs of 25 steps each. Full details on this experiment are available in Table 2 in Section C.1 of the appendix.

a)



b)

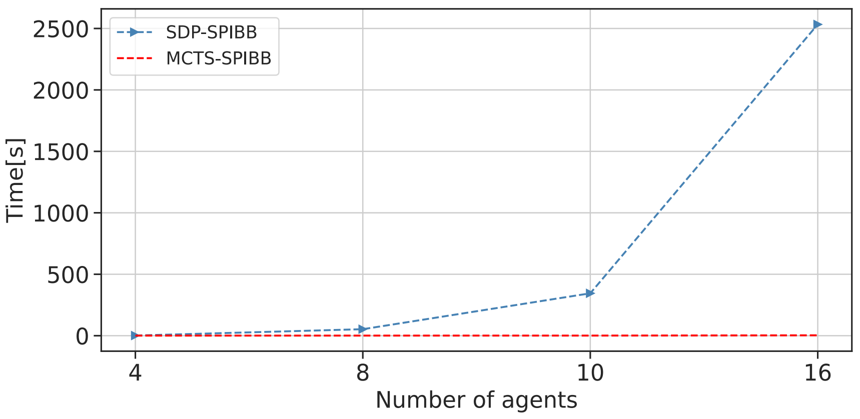


Fig. 3. Scalability on multi-agent SysAdmin: a) performance (y-axis) of the baseline policy ($p = 0.7$, yellow boxplots), SDP-SPIBB (steel-blue boxplots) and MCTS-SPIBB (red boxplots) as the number of agents varies (x-axis); b) average computational time (y-axis) of SDP-SPIBB, MCTS-SPIBB as the number of agents varies (x-axis).

In Figure 3.a, the box plots show the performance (y-axis) of the algorithms as the number of agents increases (x-axis). MCTS-SPIBB (with 10000 simulations) and SDP-SPIBB achieve similar performance and generate significant improvements over the baseline policy. In this domain, the state space and action space sizes grow exponentially with the number of agents, then the comparison is only possible between MCTS-SPIBB and SDP-SPIBB, as SPIBB, DP-SPIBB, and state-of-the-art SPI methods, based on policy iteration, are not able to scale even to the minimum number of agents of this benchmark. SPIBB and DP-SPIBB stop working with 3 agents because they are based on matrix inversion or improve all states in each iteration. In Figure 3.b, the lines show the computational times (y-axis) for the policy improvement as the number of agents (x-axis) increases. In this case, both algorithms scale as the number of agents varies, however, SDP-SPIBB begins to exhibit larger computational time compared to MCTS-SPIBB from 8 agents, due to the growth of the non-bootstrapped set $\overline{\mathcal{B}}$. The complexity of SDP-SPIBB is tied to the size of this set, as explained in Section 3.2, which usually grows with the amount of data collected. For this reason, the computation times of SDP-SPIBB can also become very high in applications where a huge amount of data is collected. Consider, for example, a real-world scenario where the application involves data streams. In such a case, the policy improvement for SDP-SPIBB requires the algorithm to consider all possible state-action pairs in $\overline{\mathcal{B}}$ that can become very large. MCTS-SPIBB is also robust to those scenarios since its complexity depends only on the number of simulations performed.

4.6.3 Performance and Scalability under Varying Baselines. We further evaluate the scalability of MCTS-SPIBB (with 10000 simulations) and SDP-SPIBB, also considering the sensitivity of the algorithms' performance to the randomness of the baseline on multi-agent SysAdmin with 16 agents. For the case with 16 agents, we generated 20 datasets of $[10000, 50000, 100000, 200000, 500000, 1700000]$ trajectories using two baseline policies: a policy (called random in the following) that selects optimal actions with probability $p = 0.5$ and random actions with the rest of probability; a policy (called sub-optimal in the following) that chooses the optimal action with probability $p = 0.9$ and uniformly random actions with the rest of the probability. The algorithms are evaluated using the average discounted return $\bar{\rho}(\pi_I, M^*) = V_{M^*}^{\pi_I}(s_0)$, over 20 runs of 25 steps each. All parameters of this experiment are available in Table 2 in Section C.1 of the appendix. In Figures 4.a and 4.c, the box plots show the performance of MCTS-SPIBB and SDP-SPIBB (y-axis) with the sub-optimal and the random baseline policy, respectively, as the dataset size increases (x-axis). With both considered baseline policies, the improvement of MCTS-SPIBB and SDP-SPIBB becomes significant starting from a sufficient number of trajectories, in this case, $|\mathcal{D}| = 50000$. In Figures 4.b and 4.d, the lines show the computational times of MCTS-SPIBB and SDP-SPIBB (y-axis), respectively, with the sub-optimal and the random baseline policy, as the dataset size increases (x-axis). Furthermore, for each point, we show the number of state-action pairs $(s, a) \in \overline{\mathcal{B}}$. SDP-SPIBB shows increased computation times as the set $\overline{\mathcal{B}}$ grows. In contrast, MCTS-SPIBB has a complexity that depends only on the number of simulations, hence it is constant in the experiment.

One of the key advantages of MCTS-SPIBB over SDP-SPIBB is its improved scalability in terms of time. This improvement is achieved because MCTS-SPIBB focuses on calculating policy improvement only in the visited states, rather than across all non-bootstrapped state-action pairs, as SDP-SPIBB does. By adopting this approach, MCTS-SPIBB significantly reduces the computational burden, resulting in more efficient computation.

4.7 Safety

In this section, we assess the safety guarantees of MCTS-SPIBB and SDP-SPIBB by comparing them with state-of-the-art Safe Policy Improvement methods, including both model-based and model-free RL approaches, with a specific focus on a detailed comparison with SPIBB.

4.7.1 Comparison with the State-of-the-Art Baseline SPI Methods. We evaluate the performance of MCTS-SPIBB and SDP-SPIBB on the *WetChicken* domain introduced as a benchmark in (Scholl et al. 2022a). Specifically, we compare our methods against a broad range of SPI baselines, including SPIBB, DUIPI, R-MIN, RaMDP, and variants such as Lower-SPIBB, Approx-Soft-SPIBB, and Adv-Approx-Soft-SPIBB. The evaluation is conducted across multiple dataset sizes, with $|\mathcal{D}| \in 100, 200, 500, 1000, 2000, 5000$. For each dataset size, we generate 20 datasets using a baseline policy that selects the optimal action with probability $p = 0.7$. Performance is measured

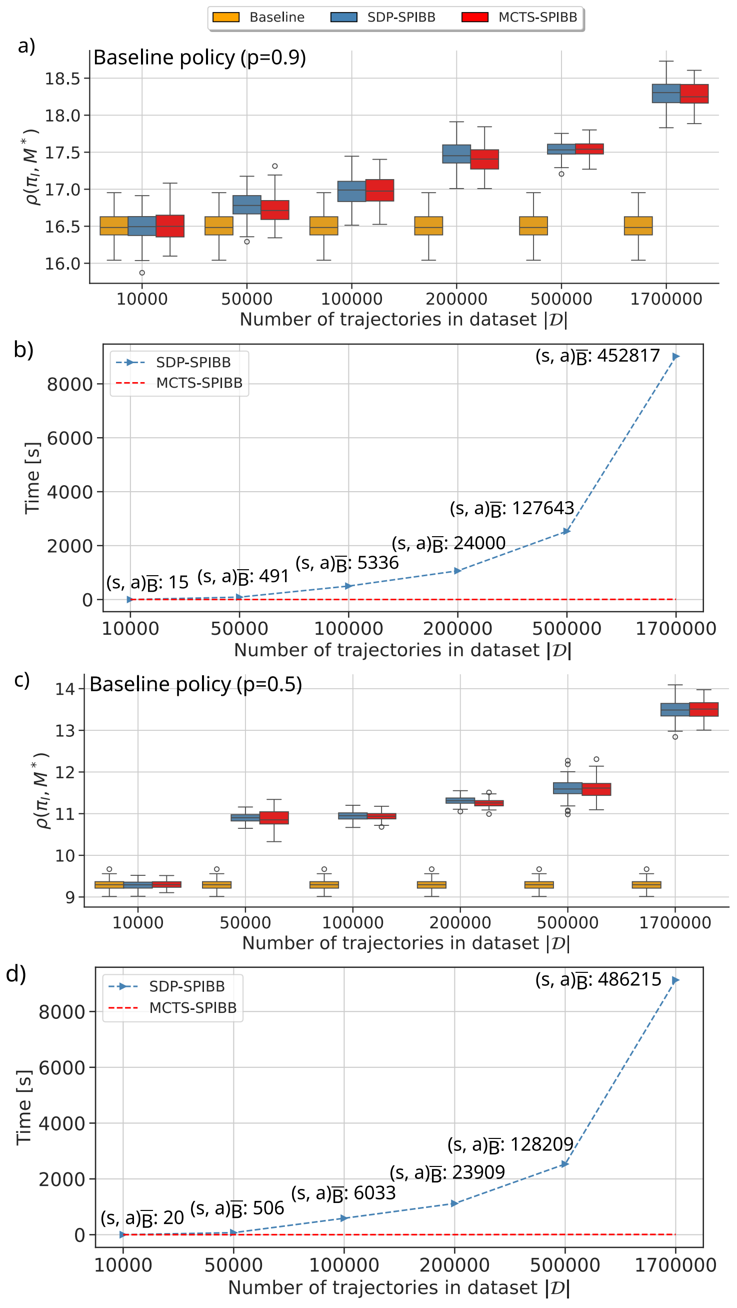


Fig. 4. Sensitivity of MCTS-SPIBB and SDP-SPIBB to baseline policy quality and dataset size in the multi-agent SysAdmin domain:(a,c) performance with sub-optimal ($p = 0.9$) and random ($p = 0.5$) baselines; (b,d) computational time vs. dataset size. Non-bootstrapped state-action counts are also reported.

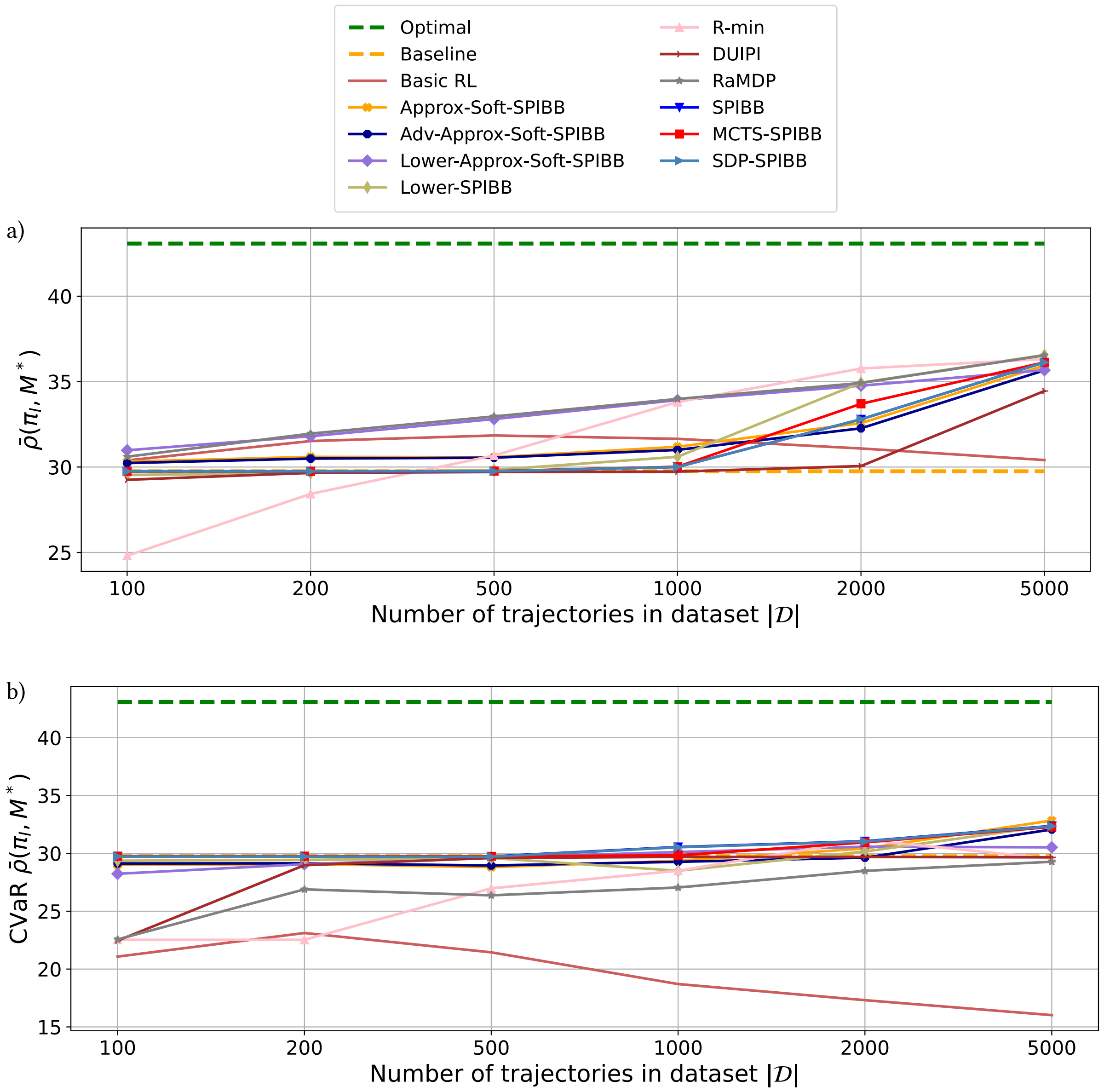


Fig. 5. Safety of MCTS-SPIBB, SDP-SPIBB and state-of-the-art SPI algorithms on *WetChicken*: a) performance (y-axis) as the dataset size varies (x-axis) of the optimal policy (green dashed line), sub-optimal baseline policy ($p = 0.7$, yellow dashed line), SDP-SPIBB (steel-blue line with right-triangle marker), SPIBB (blue line with downward-triangle marker), MCTS-SPIBB (red line with square marker) and the others state-of-the-art SPI algorithms of the benchmark proposed in (Scholl et al. 2022a); b) 15% CVaR (y-axis) of the algorithms as the dataset size increases (x-axis).

via exact policy evaluation on the true MDP M^*. Complete details of the experimental setup can be found in Table 3, Section C.1. Figures 5.a and 5.b show the mean return and the 15% Conditional Value at Risk (CVaR), respectively, as a function of dataset size. As shown in (Scholl et al. 2022a), not all evaluated methods are provably safe: only SPIBB, DUIPI, and Adv-Approx-Soft-SPIBB provide both theoretical and empirical safety guarantees. Building on these results, we observe that MCTS-SPIBB and SDP-SPIBB consistently achieve state-of-the-art performance in terms of both average and worst-case performance, thereby validating their safety properties from both theoretical and empirical perspectives. Basic-RL consistently underperforms compared to SPI methods, in terms of both mean and CVaR. Notably, its CVaR performance, which captures worst-case outcomes, deteriorates as the number of trajectories increases, further emphasizing the safety advantage of SPI algorithms, which becomes particularly evident when uncertainty in the action-value function is considered. This degradation occurs because Basic RL increasingly exploits the MLE model as more data become available, without any safety constraints. Such exploitation amplifies the effect of model inaccuracies in underrepresented regions, leading to poorer performance. Interestingly, this consideration appears to be beneficial not only for CVaR, but also for mean performance, which improves as a side effect of safer policy updates. This behavior illustrates the risks of relying solely on model accuracy without enforcing safety constraints and aligns with the empirical findings reported in (Scholl et al. 2022a). The only difference is that, while their experiments extend to a larger number of trajectories, we limit our analysis up to 5000. This choice is motivated by the observation that the most interesting differences across methods emerge in the low-data regime (e.g., 100 to 5000 trajectories), where safety guarantees are most critical. Beyond this threshold, our experiments indicate that results tend to converge and closely match those reported in (Scholl et al. 2022a), thus providing limited additional insight. As shown in Figure 5 (page 22) in (Scholl et al. 2022a), Basic RL even begins to recover in terms of mean performance as the data increases, yet its CVaR continues to degrade, highlighting the trade-off between average return and safety in the absence of constrained policy updates.

4.7.2 Comparison with Model-Free/Model-Based Offline RL Methods. To further contextualize our methods, we include several representative model-free and model-based offline RL algorithms into the same WetChicken-based setup used for SPI evaluation. Specifically, we consider DQN (Mnih et al. 2013), PPO (Schulman et al. 2017), CQL (Kumar, Zhou, et al. 2020), Decision Transformer (Chen et al. 2021), and MOReL (Kidambi et al. 2020). To ensure fair comparisons with our SPIBB-based methods using the same data and environment models, and to assess how standard model-free algorithms behave under purely offline constraints, we trained these algorithms solely on offline data using the MLE-estimated transition model obtained from the dataset as a simulator of the environment when needed, except for MOReL. Since MOReL is a model-based offline RL method, it learns the dynamics via an ensemble of neural networks, hence, we only supplied it with data collected by executing the baseline policy in the real environment. We performed a grid search over hyperparameters (as detailed in Table 3), using 10 independent runs for each configuration. For evaluation, we selected the best-performing configuration for each method, which is highlighted in bold in Table 3. Figures 6.a and 6.b report the mean return and the 15% CVaR, respectively. The results show that none of the additional methods provides stable or reliable performance under limited data conditions. In particular, we observe high variance and policy degradation in low-data regimes, confirming that optimizing policies over poorly supported models, without incorporating safety constraints, can lead to unreliable behavior, namely, a performance decrease. CQL achieves higher and slightly more stable performance compared to the other additional methods, but it still fails to guarantee performance that match or surpass the baseline policy when only a few trajectories are available. To better isolate the role of model bias and confirm whether performance degradation in PPO, DQN, and CQL stems from uncertainty in the estimated dynamics, we conducted an additional experiment in which the MLE model is replaced by the true transition model. In this idealized scenario, free from model error, all three algorithms reliably converge to the optimal policy's performance, as expected. This confirms that their instability in low-data settings is not due to

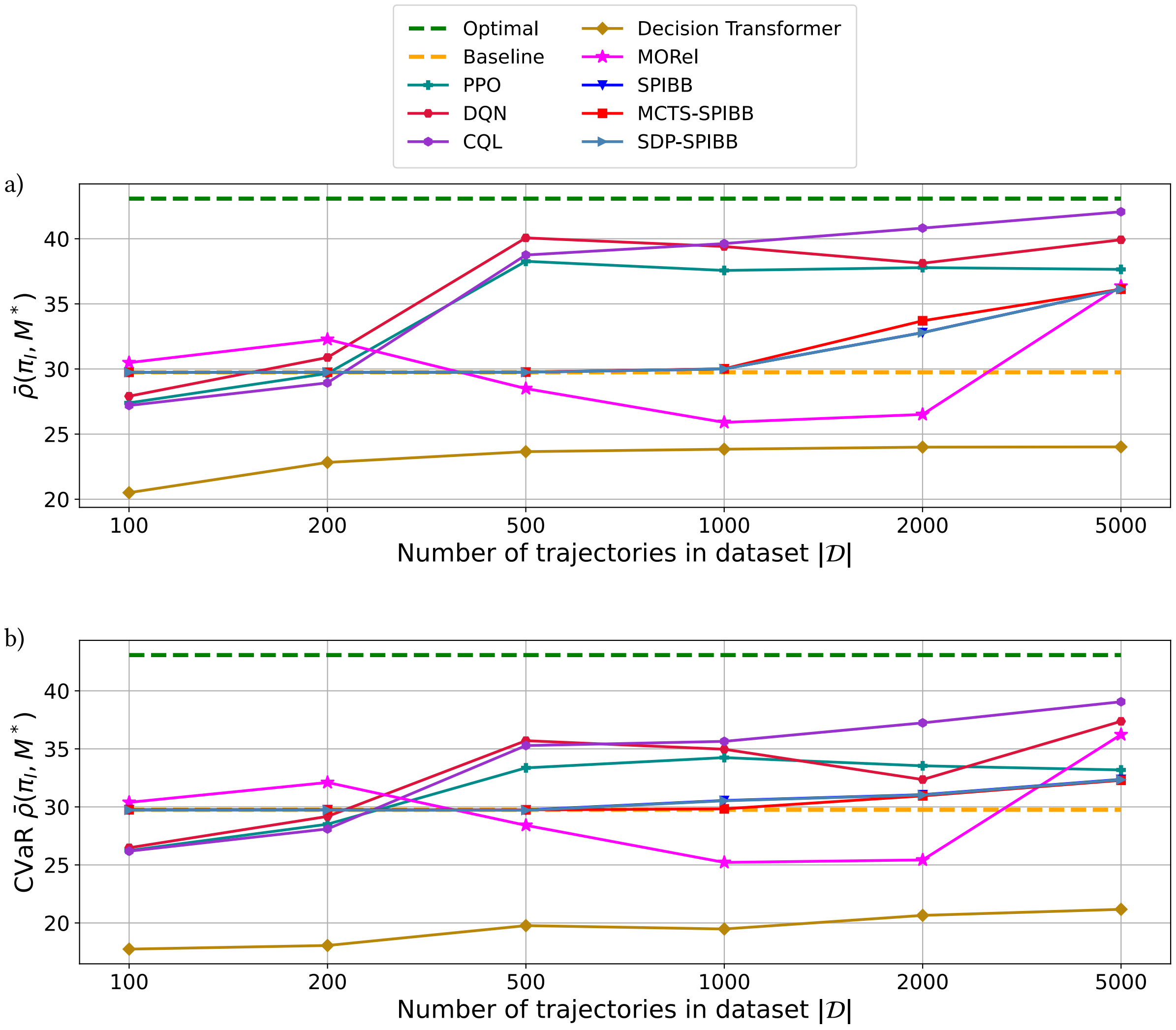


Fig. 6. Safety of MCTS-SPIBB, SDP-SPIBB and state-of-the-art SPI algorithms on *WetChicken*: a) performance (y-axis) as the dataset size varies (x-axis) of the optimal policy (green dashed line), sub-optimal baseline policy ($p = 0.7$, yellow dashed line), SDP-SPIBB (steel-blue line with right-triangle marker), SPIBB (blue line with downward-triangle marker), MCTS-SPIBB (red line with square marker), Basic-RL (iron oxide red line), PPO (teal line with plus markers), DQN (dark-red line with hexagon markers), CQL (purple line with hexagon markers), Decision Transformer (dark goldenrod line with diamond markers) and MOReL (magenta line with star markers); b) 15% CVaR (y-axis) of the algorithms as the dataset size increases (x-axis).

inherent algorithmic limitations, but rather to their sensitivity to model inaccuracies in the absence of explicit safety constraints. Decision Transformer behaves differently, even in this favorable setting, it fails to reach the performance level of the other evaluated methods.

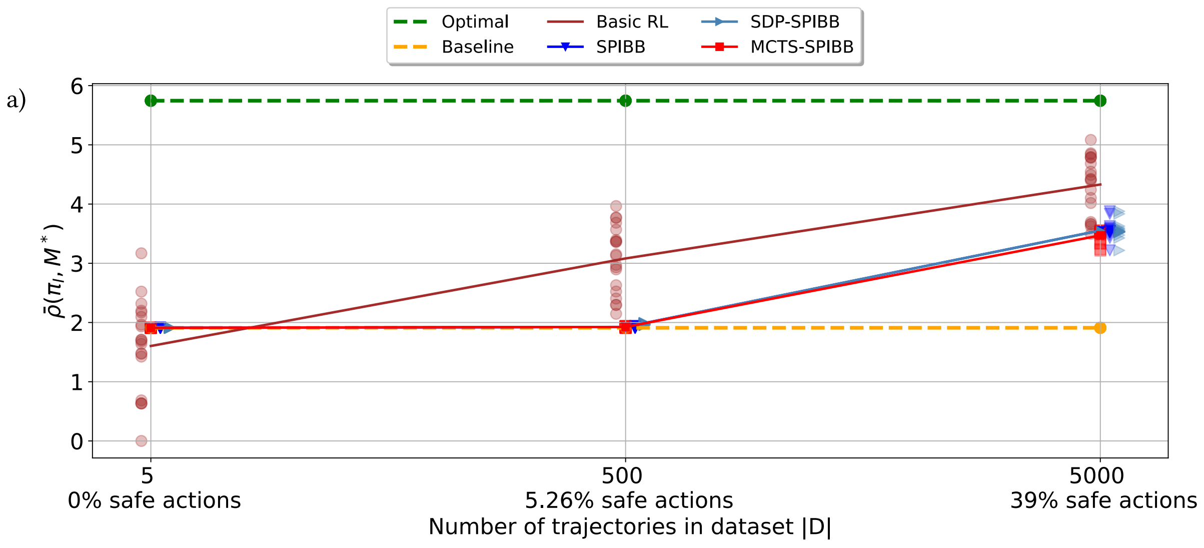
Regarding MOReL, our experiments show that it requires extensive tuning, and the optimal hyperparameter configuration found for one dataset size does not generalize well to others. As seen in Figures 6.a and 6.b, the method shows significant instability at 500, 1000, and 2000 trajectories, with performance only stabilizing around 5000. This volatility is primarily due to the absence of baseline-related guarantees. With small datasets, the ensemble variance is low, resulting in minimal penalties for uncertain actions. As the dataset size grows, but before the model is sufficiently accurate, the variance increases, making MOReL overly conservative. Crucially, MOReL and SPIBB-based methods differ in the type of guarantees they provide. SPIBB-based approaches ensure that the learned policy will not perform worse than the baseline, leveraging a tabular MLE model and count-based thresholds to enforce conservative updates. These yield rigorous PAC-style guarantees that hold with high probability in the true MDP. In contrast, MOReL aims to establish a global lower bound on return in the true MDP by penalizing uncertain regions through a pessimistic model (P-MDP). While its theoretical formulation relies on a known total variation bound α, the practical implementation estimates uncertainty via the variance of an ensemble of neural networks. This provides empirical robustness, but lacks the formal guarantees of SPIBB, particularly in low-data regimes. Moreover, MOReL does not ensure baseline-relative safety, which remains the core principle behind SPIBB and our proposed methods. In summary, our analysis reveals a fundamental trade-off between empirical performance and theoretical reliability. SPIBB-based methods, including MCTS-SPIBB and SDP-SPIBB, enforce conservative updates grounded in data support and, as a result, maintain reliable performance even under limited data conditions. Specifically, the way uncertainty is handled in our framework favors safety over aggressive policy improvement. While this may lead to lower mean performance compared to other methods in high-data regimes, it ensures far greater stability and reliable policy, particularly in safety-critical or data-scarce environments.

4.7.3 In-Depth Comparison with SPIBB. We focus the comparison on safety with MCTS-SPIBB, SPIBB, SDP-SPIBB, and Basic RL (Basic RL is the vanilla Offline RL used as a non-safe baseline also by (Laroche et al. 2019)). We considered the benchmark presented by (Castellini et al. 2023; Laroche et al. 2019), namely, SysAdmin with 7 machines and Gridworld 5×5, varying the dataset size. In SysAdmin, the dataset sizes considered are $|\mathcal{D}| = [5, 500, 5000]$; for Gridworld $|\mathcal{D}| = [2, 10, 100, 1000, 10000]$. For each case, i.e., for each dataset size, we generated 20 datasets using a baseline policy that selects the optimal action with a probability $p = 0.7$. The performance of the algorithms is computed in each case using Policy Evaluation. All parameters of this experiment are available in Table 4 in Section C.1 in the appendix.

In Figures 7.a and 7.b, the lines show respectively the average performance and the 15% Conditional Value at Risk (CVaR) (y-axis) of the algorithms on single-agent SysAdmin with 7 machines as the dataset size increases (x-axis). Figures 8.a and 8.b, show the same but the evaluation of the algorithms is made on 5×5 GridWorld. These experiments are an extension of the results obtained in the benchmark proposed in (Castellini et al. 2023), to which the evaluation of MCTS-SPIBB and SDP-SPIBB has been added. The results confirm that the performance of the policy computed by MCTS-SPIBB converges to that of SPIBB and that SDP-SPIBB has performance equivalent to SPIBB. Consequently, MCTS-SPIBB and SDP-SPIBB are safe algorithms with state-of-the-art performance.

4.8 Ablation Study

In this section, we present a set of ablation studies designed to better understand the behavior of our proposed algorithms, MCTS-SPIBB and SDP-SPIBB, under varying dataset and algorithmic conditions. While our main experiments focus on benchmarking performance across different baselines and domains, here we delve deeper into how key factors such as dataset size, baseline policy quality, and simulation budget affect the safety and effectiveness of policy improvement. We first evaluate how the number of available trajectories impacts performance when using a random baseline policy, which provides broad, but noisy, coverage of the environment. Next, we analyze how the diversity and quality of the baseline policy influence dataset coverage and the resulting safe improvements, contrasting random and sub-optimal policies. Finally, we investigate the sensitivity of MCTS-SPIBB to the number of Monte Carlo simulations to assess how approximation quality affects empirical



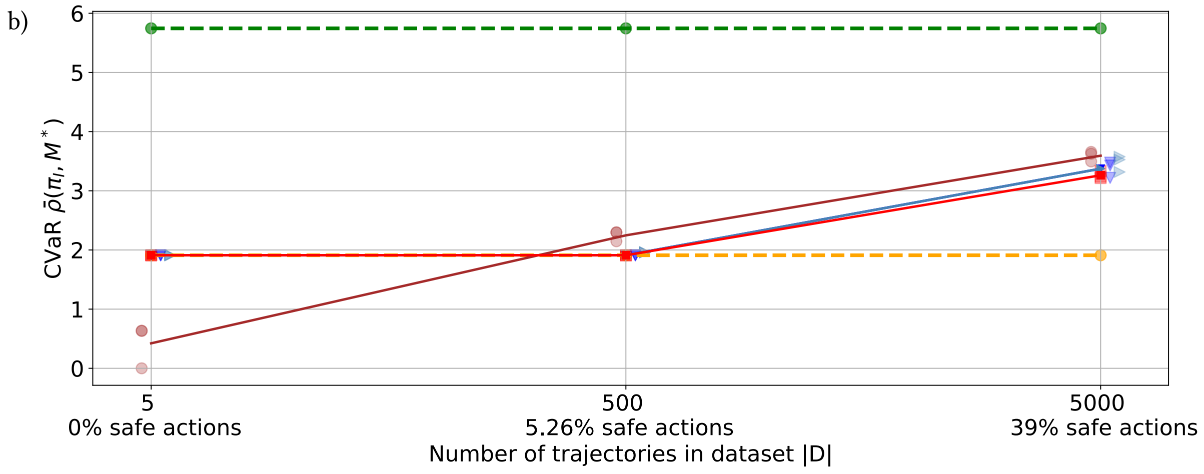


Fig. 7. Results about safety on SysAdmin with 7 machines. Performance $\rho(\pi_I, M^*)$ and 15%-CVaR $\rho(\pi_I, M^*)$ (y-axis) depending on dataset size $\mathcal{D}$ (x-axis).

robustness. These analyses provide insight into the mechanisms driving performance in our methods and help identify conditions under which safety guarantees are preserved or challenged.

4.8.1 Performance Sensitivity to Dataset Size under a Random Baseline. We compare the sensitivity of the performance of MCTS-SPIBB, SDP-SPIBB, and SPIBB to the number of trajectories in the dataset on the random baseline policy. The baseline policy chooses actions randomly with a probability of 0.5. The evaluation is conducted on the SysAdmin domain with 7 machines, focusing on the capability of both algorithms to generate safe policy improvements varying the size of the dataset, i.e., $|\mathcal{D}| = [5, 100, 1000, 5000, 10000, 100000]$, but starting with a poor quality baseline policy. For each case, i.e., for each dataset size, we generated 20 datasets with random policy.

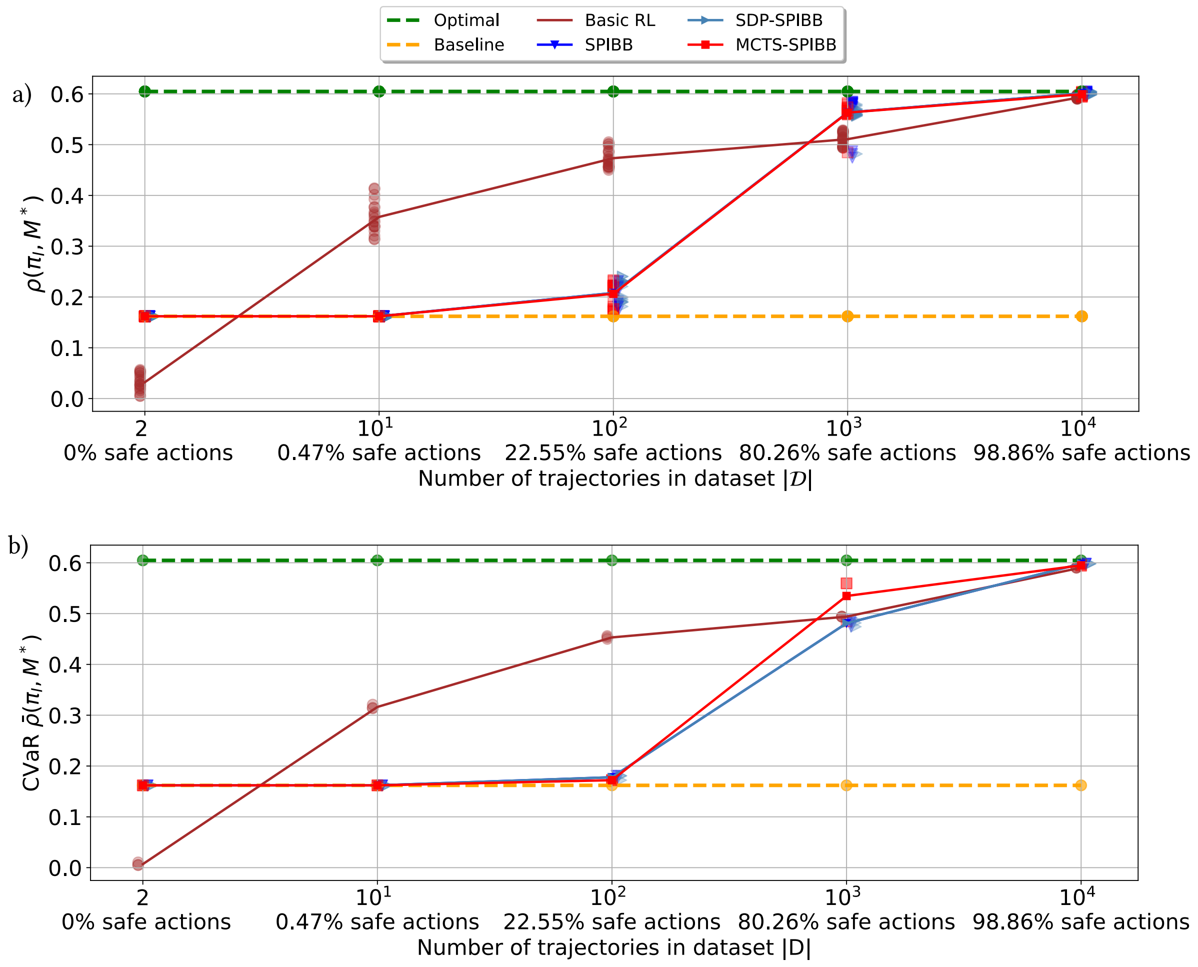


Fig. 8. Results about safety on GridWorld. Performance $\rho(\pi_I, M^*)$ and 15%-CVaR $\rho(\pi_I, M^*)$ (y-axis) depending on dataset size $\mathcal{D}$ (x-axis).

The algorithms are evaluated in each case using Policy Evaluation. Full details on this experiment are available in Table 4 in Section C.1 in the appendix.

In Figure 9, the lines show the average performance (y-axis) of SDP-SPIBB, MCTS-SPIBB, and SPIBB, as the number of trajectories increases (x-axis) and a starting random baseline policy (yellow dashed line) is used for collecting the dataset. The results show that with a few trajectories, all the algorithms have performance similar to the baseline policy, as expected. The performance improvement becomes more significant as the dataset size increases for both algorithms, converging to performance close to the optimal policy (green dashed line). Notice that, when the transition model representation is inaccurate, such as when the dataset trajectories are few, the performance of the algorithms tends to degrade. The fact that MCTS-SPIBB, SDP-SPIBB, and SPIBB maintain

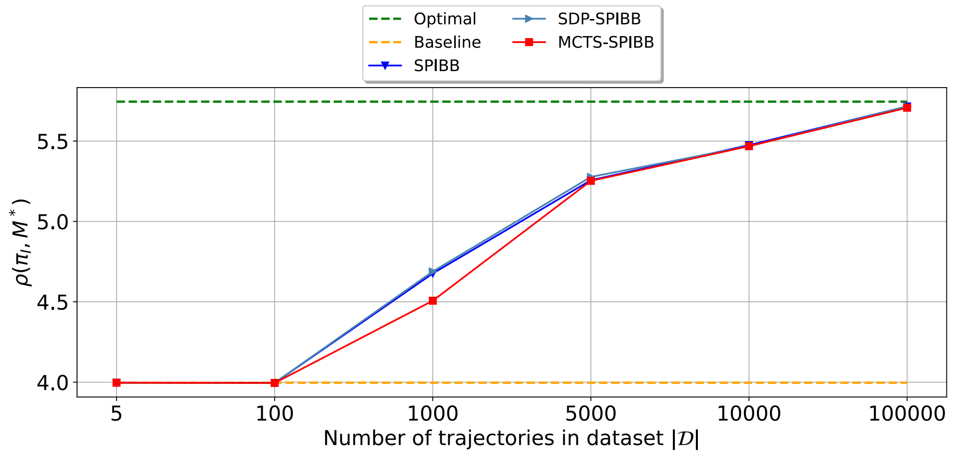


Fig. 9. Sensitivity of performance to the number of trajectories and randomness of baseline policy on SysAdmin with 7 agents. Performance $\rho(\pi_I, M^*)$ (y-axis) depending on dataset size $\mathcal{D}$ (x-axis) and the random baseline policy ($p = 0.5$).

performance equal to or better than the baseline policy with few trajectories is a positive factor, as the safety guarantees hold also in those extreme cases.

4.8.2 Sensitivity to Baseline Policy and Dataset Coverage. This experiment is particularly useful for evaluating the impact that datasets collected from different baseline policies have on performance. Specifically, a random baseline policy can induce greater exploration of the state and action space, leading to datasets that contain many more diverse state-action pairs and non-optimal execution trajectories. This could be an advantage in terms of exploration, but also a disadvantage for the SPI algorithms that need enough (i.e., more than $N_\wedge$) occurrences of state-action pairs to reduce the uncertainty on the transition model and improve the policy. These difficulties are more likely in large-scale domains where the number of states and actions is exponentially high. On the other hand, a sub-optimal baseline policy can reduce the explored space by collecting only optimal execution trajectories, but at the same time, it can increase the number of state-action pairs observed enough (i.e., more than $N_\wedge$) times, which can be used by the SPI algorithm to perform the improvement.

In this experiment, the random and sub-optimal policies (described above) are used on the SysAdmin domain with 7 machines. We assess the capability of both MCTS-SPIBB (with 10000 simulations), SDP-SPIBB, and SPIBB, to generate safe policy improvements over each baseline policy. The box plots in Figure 10 show the average performance (y-axis) of SDP-SPIBB, MCTS-SPIBB, and SPIBB, as different baseline policies are used for collecting data (x-axis). All parameters of this experiment are available in Table 5 in Section C.1 of the appendix. In both cases, MCTS-SPIBB, SDP-SPIBB, and SPIBB, achieve better performance than the reference baseline policy and perform close to the optimal policy. In the first case, where the baseline policy is random, MCTS-SPIBB appears to achieve slightly lower performance, and this is influenced by the number of simulations used in this experiment. A higher number of simulations would allow for the calculation of a policy with better performance, which would further minimize this difference.

4.8.3 Robustness of MCTS-SPIBB to Simulation Budget Variation. We evaluate the sensitivity of MCTS-SPIBB to the number of simulations, considering two baseline policies, namely, a random policy and a sub-optimal policy. Experiments are conducted on the *SysAdmin* domain with 7 machines, using MCTS-SPIBB with simulation

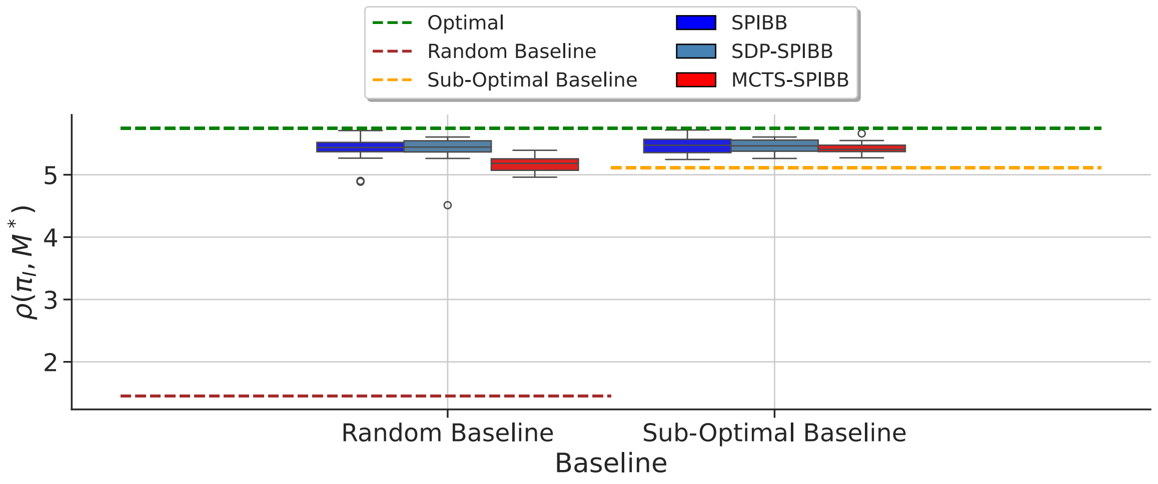


Fig. 10. Sensitivity of the performance (y-axis) of MCTS-SPIBB (red box-plots), SDP-SPIBB (steel-blue box-plots), and SPIBB (blue box-plots) to the diversity of the dataset generated by a random baseline policy ($p = 0.5$, brown dashed line) or by a sub-optimal baseline policy ($p = 0.9$, yellow dashed line) (x-axis) on SysAdmin with 7 agents.

budgets $m \in 100, 1000, 10000$ and 5000 trajectories. Full experimental details are provided in Table 5, Section C.1 in the appendix.

Figures 11.a and 11.b show the average performance (y-axis) of MCTS-SPIBB as a function of the number of simulations (x-axis), for datasets collected with the sub-optimal and random baselines, respectively. The results show that MCTS-SPIBB robustly improves over the baseline policy in both settings, and that increasing the number of simulations consistently enhances performance. Importantly, the improved policy quickly outperforms the initial baseline, even under relatively low simulation budgets. When the dataset is generated by the sub-optimal policy, the learned policy tends to achieve returns that are closer to the optimal policy than those obtained under the random baseline. This reflects the stronger initial guidance provided by the baseline in well-supported regions of the state space, which allows MCTS-SPIBB to focus its exploration and benefit earlier from safe updates. In contrast, when the baseline is random, more simulations are required to achieve the same level of performance, due to the broader uncertainty in value estimation. These results empirically support the robustness of MCTS-SPIBB under computational constraints. While the use of sampled rollouts, rather than exact dynamic programming, introduces an approximation layer that may, in principle, affect the theoretical guarantees when the simulation budget m is small, MCTS-SPIBB mitigates this limitation by enforcing SPIBB constraints directly during tree expansion.

4.9 Discussion

In this section, we reflect on the algorithmic foundations, design trade-offs, and empirical insights gained from the proposed methods. We begin by highlighting the conceptual similarities and differences between MCTS-SPIBB and SDP-SPIBB, which approach the problem of safe policy improvement from two complementary perspectives. We then analyze the robustness of MCTS-SPIBB under limited simulation budgets and discuss the practical

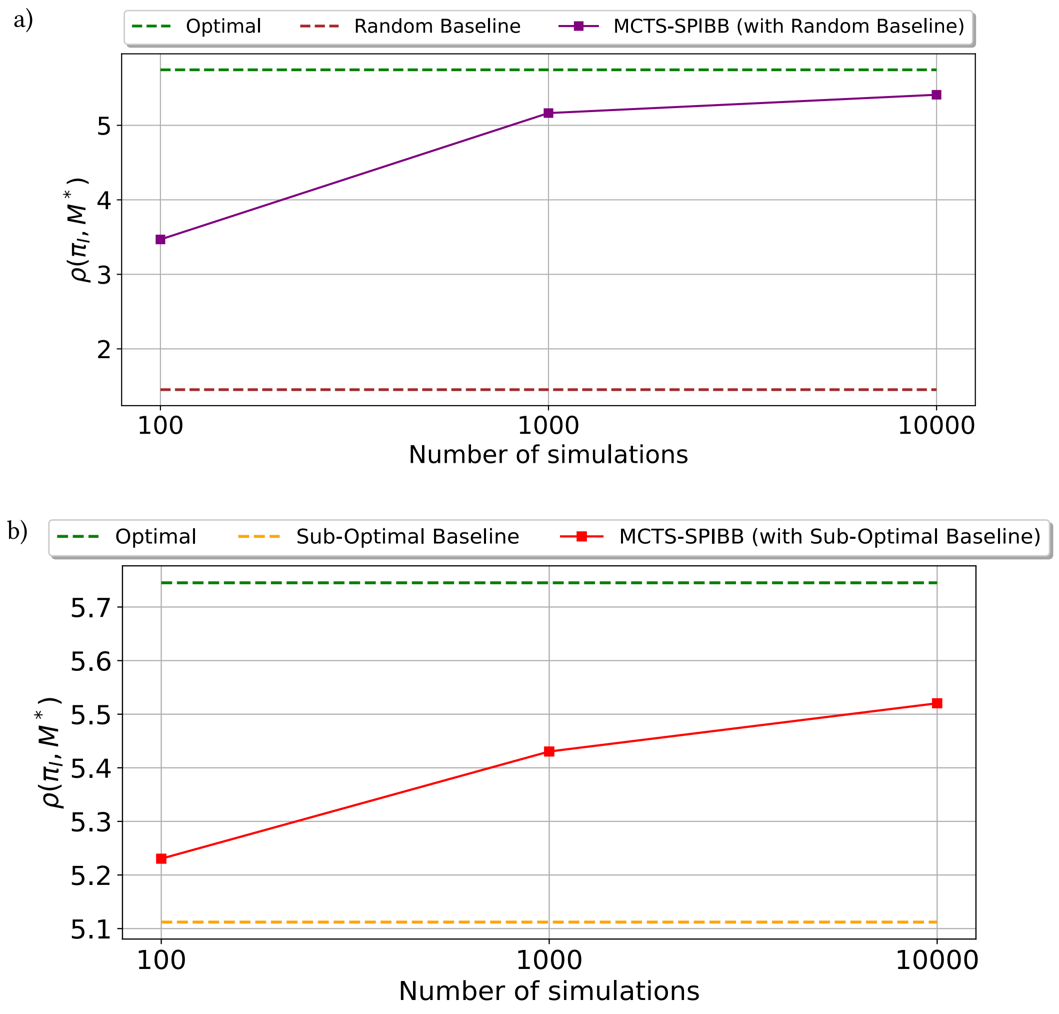


Fig. 11. Sensitivity of performance (y-axis) of MCTS-SPIBB as the number of simulations increases (x-axis) on SysAdmin with 7 agents and a random baseline policy (a) or a sub-optimal policy (b) is considered.

complexity of SDP-SPIBB in sparse, large-scale environments. This analysis sheds light on the scalability and reliability of both approaches in different offline RL scenarios.

4.9.1 Connections between MCTS-SPIBB and SDP-SPIBB. Although MCTS-SPIBB and SDP-SPIBB are built upon distinct algorithmic foundations, both methods are unified by a common goal, making safe policy improvement under SPIBB constraints scalable to large discrete MDPs. However, the two approaches tackle the challenge from different perspectives: MCTS-SPIBB addresses scalability by computing the policy locally, using a sampling-based strategy integrated into an original SPIBB-constrained search tree; SDP-SPIBB, instead, needs to represent the entire policy (which is difficult in very large environments) and accelerates the dynamic programming-based policy iteration by focusing the policy evaluation only on transactions supported by data. Analyzing in depth the two approaches, we see that SDP-SPIBB improves the baseline policy performing a cycle on all the states-action pairs with enough data support. For each of these pairs, it computes Q-values, and from Q-values, the improved policy. SDP-SPIBB has a global view over all states (also those that are very far away from the current state of the agent) because dynamic programming approaches are intrinsically offline and global w.r.t. the state space. The complexity of SDP-SPIBB depends on the number of non-bootstrapped state-action pairs $|\bar{\mathcal{B}}|$. On the other hand, MCTS-based methods compute Q-values locally and online, focusing only on states that are close enough to the agent to affect its current behavior. Thus, MCTS incrementally builds a partial tree search and estimates Q-values only for the subset of the state space encountered during the simulations. MCTS-SPIBB inherits this strategy and computes the Q-values and the corresponding improved policy only for the current state, which makes a big difference in huge environments. This approach, in fact, allows the method to scale better to large domains, as it avoids computing values for states that are irrelevant to the current decision. The policy improvement strategy implemented by MCTS-SPIBB makes its computational complexity dependent on the number of simulations, instead of on the state space dimension (as in state-of-the-art SPIBB) or the dataset dimension.

To summarize: both SDP-SPIBB and MCTS-SPIBB are valid ways to compute Q-values within the SPIBB framework, and our contributions show how these two strategies can be effectively adapted to enforce safe policy improvement while addressing scalability from different angles.

4.9.2 Robustness of MCTS-SPIBB under Limited Simulation Budgets. A key consideration in the design of MCTS-SPIBB is the use of sampling-based value estimation. Unlike exact dynamic programming methods, MCTS-based approaches construct value estimates incrementally through stochastic rollouts. This introduces an additional approximation layer that can, in principle, affect the value estimates and compromise the theoretical guarantees when the number of simulations is small. This limitation is not specific to MCTS-SPIBB, but is inherent to all MCTS-based approaches, where value estimates are constructed incrementally from stochastic simulations. To mitigate this, MCTS-SPIBB integrates the SPIBB constraints directly into the tree expansion, ensuring that policy updates remain conservative and limited to regions with sufficient data support. The empirical sensitivity analysis across varying values of m (see Section 4.8.3) shows that MCTS-SPIBB performs robustly even with relatively low simulation budgets. Our further experimental evaluation (see Section 4.6.2) shows that, in complex multi-agent environments where the state and action spaces grow exponentially with the number of agents n, with a state space size of $|S| = 9^n$ and an action space size of $|A| = 2^n$, both safety and performance improvement are achieved even with relatively small values of m. While a full theoretical analysis of the trade-off between m, approximation bias, and safety remains an open challenge, we consider this an important direction for future work. In particular, developing a bound on the error introduced by finite rollouts could help formalize guarantees for sampling-based safe planning methods.

4.9.3 Complexity of SDP-SPIBB. In the worst-case scenario, where all states and actions are sufficiently supported by collected data, the complexity of SDP-SPIBB can approach that of standard DP-SPIBB, namely $O(|S|^2|A|^2)$. However, in large environments, it is usually impossible to collect a dataset that supports all state-action pairs, namely, it would take too much time for the baseline policy. In these cases, which happen often in real-world settings, the offline dataset is highly sparse and concentrated on a relatively small subset of transitions. As a result, the effective complexity, which depends on $|\bar{\mathcal{B}}|$ is typically several orders of magnitude smaller than $|S|^2|A|^2$. Specifically, in the experiment from Section 4.6.2, shown in Figures 4.b,d, we observed that $|\bar{\mathcal{B}}|$ grows with dataset size, reflecting the expected behavior, as more transitions surpass the safety threshold, more updates become eligible for improvement. For example, with 1700000 trajectories, $|\bar{\mathcal{B}}| = 452817$, which represents the number of state-action pairs that are sufficiently supported to allow safe policy improvement. While this number may appear large, it is much smaller than the total number of possible state-action pairs in the domain with 16 agents which is $\approx 10^{20}$ since $|S| = 9^{16} \approx 10^{15}$, $|A| = 2^{16} = 65536 \approx 10^5$). Another example, in which we have a comparison with SPIBB (see Figure 7 in Section 4.6.1), is the SysAdmin domain with 7 machines: the total number of possible state-action-next-state combinations is on the order of $|S|^2|A|^2 = 128^2 \cdot 2^2 = 65536$. However, even with 5000 trajectories, the non-bootstrapped set $\bar{\mathcal{B}}$ remains around 1900 entries, highlighting the practical gap between worst-case and observed complexity. Moreover, both SDP-SPIBB and MCTS-SPIBB converge to the performance of the policy computed by SPIBB.

It is also important to clarify the distinction between the two experimental setups. Section 4.6.1 evaluates performance in a single-agent setting, where the x-axis represents the number of machines controlled by a single agent. In this case, datasets are relatively small, and SDP-SPIBB scales significantly better than baseline methods, which struggle with more than $10 - 12$ machines. This setup tests scalability with respect to problem size, i.e., the size of the state and action spaces. In contrast, Figure 4 in Section 4.6.2 refers to a multi-agent setting, using extremely large datasets ($|\mathcal{D}| \in 10000, 50000, 100000, 200000, 500000, 1700000$). Here, we stress-test both SDP-SPIBB and MCTS-SPIBB under high data volume conditions. In this scenario, MCTS-SPIBB scales independently of dataset size, while SDP-SPIBB scales independently of state space size, though its complexity grows with the number of safe transitions. In contrast, SPIBB and DP-SPIBB are not executed in this domain, as their reliance on full dynamic programming renders them infeasible in large-scale environments. This limitation highlights a key advantage of SDP-SPIBB, it can safely improve the policy and remains applicable in high-dimensional settings. While its runtime increases with dataset size, this growth reflects the algorithm's ability to safely leverage larger datasets, rather than a fundamental computational bottleneck. As shown in Figures 4.b,d, SDP-SPIBB scales effectively across all tested settings, making it a practical solution for offline RL in complex domains.

4.9.4 The Role of Bootstrapped Sets in Preserving Safety. Certain ablations, such as applying UCT uniformly to both bootstrapped and non-bootstrapped sets, or excluding all state–action pairs in the bootstrapped set, are not meaningful within the SPIBB framework. The safety guarantees of SPIBB rely precisely on treating these sets differently, for the bootstrapped set, the policy must remain close to the baseline to ensure safety. Applying UCT without this distinction would remove the constraints, reducing the method to standard MCTS and breaking the theoretical guarantees. Moreover, if the policy updates are applied directly to the MLE MDP without SPIBB constraints, the algorithm would rely more heavily on the model estimates. With limited data, these estimates may be inaccurate, and unconstrained updates on poorly supported state–action pairs could degrade performance relative to the baseline policy. Similarly, excluding all state–action pairs in the bootstrapped set may lead to blocking situations, since in some states all actions may be bootstrapped and thus no action would be available. In both cases, safety guarantees cannot be ensured, which is why we do not consider these ablations in our experiments.

5 Related Work

The main research topics related to our work are Offline RL, also known as Batch RL, safe policy improvement, and MCTS-based planning/RL.

5.1 Offline RL

Offline RL refers to a framework where the learning process is performed using a fixed set of data collected in a dataset. Several works provide a detailed analysis of the theoretical properties and practical implementations (Lange et al. 2012). In particular, Offline RL focuses on developing algorithms that derive control policies from data collected by a baseline policy, without additional online interaction. An overview of the main challenges and opportunities in offline RL is presented in (Levine et al. 2020). Early methods explored regression-based approaches, such as the use of regression trees to approximate the Q-function from trajectory datasets and support batch updates (Ernst et al. 2005). More recent work has focused on off-policy deep RL techniques to improve learning stability in the absence of exploration (Fujimoto et al. 2019; Kumar, Fu, et al. 2019). These methods introduce mechanisms to reduce extrapolation errors and stabilize Q-learning via bootstrapping error reduction. Conservative Q-Learning (Kumar, Zhou, et al. 2020) further builds on this direction by proposing a conservative Q-function estimation to ensure safer policy updates. However, none of these approaches explicitly incorporates the baseline policy into the learning process or provides guarantees of improvement over it.

5.2 Safe Policy Improvement

Safe policy improvement is a specialization of offline RL designed for reliable applications where a new policy must outperform a given baseline, with formal guarantees on its performance (Scholl et al. 2022a). Early work in this direction introduced percentile-based optimization techniques for MDPs with parameter uncertainty, aiming to maximize expected return in worst-case scenarios (Delage and Mannor 2010). High-confidence improvement methods were then proposed to ensure, with high probability, that the learned policy performs better than the baseline (Thomas et al. 2015). These ideas are later extended by robust baseline regret minimization techniques (Petrik et al. 2016), which aim to produce policies that are at least as good as a baseline under worst-case model realizations. More recent methods, such as SPIBB (Laroche et al. 2019), introduced constraints on modifying the baseline policy based on the empirical support of state-action pairs. Further extensions like Soft-SPIBB (Nadjahi et al. 2019) relaxed these constraints, allowing broader changes while still respecting uncertainty-based limits. Other works explored SPI in structured environments, such as factored MDPs, to improve sample efficiency (Simão and Spaan 2019). A comprehensive classification of SPI methods is provided in (Scholl et al. 2022a), which organizes them based on how uncertainty is handled. The first category includes methods that adjust the action-value function to reduce the value of uncertain actions—this group includes RaMDP (Petrik et al. 2016), DUIPI (Schneegass, Udluft, et al. 2008), and R-Min (Brafman and Tennenholtz 2003). The second category consists of methods that constrain the policy space itself, such as HCPI (Thomas et al. 2015), SPIBB and its extensions (Laroche et al. 2019; Nadjahi et al. 2019; Scholl et al. 2022b; Simão and Spaan 2019). While these approaches represent major steps forward in safe decision-making, they remain computationally expensive, especially in large-scale environments. Notably, no existing SPI method incorporates MCTS or scales effectively to high-dimensional domains. This makes MCTS-SPIBB and SDP-SPIBB the first SPI algorithms that integrate MCTS and are capable of scaling to large environments.

5.3 MCTS-Based Planning/RL

MCTS is a powerful class of algorithms designed to scale sequential decision-making in MDPs and POMDPs to large or complex domains using sampling-based planning. Early work introduced sparse-sampling algorithms for online planning that are independent of state-space dimensionality (Kearns et al. 2002). MCTS builds on this foundation, notably through the UCT algorithm (Browne et al. 2012; Chaslot et al. 2008; Coulom 2007; Kocsis and Szepesvári 2006), which balances exploration and exploitation via upper confidence bounds. UCT generalizes the UCB strategy originally developed for multi-armed bandits (Auer et al. 2002), adapting it to non-stationary settings where action values evolve over time. MCTS methods have also been extended to handle partial observability (Silver and Veness 2010), and further enhancements have explored model learning (Katt et al. 2019, 2017) and the incorporation of prior knowledge (Castellini et al. 2019; Mazzi, Castellini, et al. 2021a,b; Mazzi, Meli, et al. 2023).

One of the most notable applications of MCTS is in AlphaGo (Silver, Huang, et al. 2016), where MCTS is combined with deep neural networks to evaluate positions and simulate games of Go. This combination led to the first AI to defeat a human professional player in the game of Go. None of these algorithms, however, aims to solve the SPI problem. The challenge in this context is to consider the baseline policy inside the MCTS, which needs to extend the action selection strategy (UCT and rollout) to guarantee the safety of the improvement.

6 Conclusion and Future Work

This paper presents two novel Safe Policy Improvement approaches for large-scale problems. The proposed methods, MCTS-SPIBB and SDP-SPIBB, demonstrate significant improvements in scalability while maintaining safety guarantees compared to state-of-the-art SPI algorithms. Empirical results show that both methods outperform existing SPI algorithms in terms of scalability and safety across various domains, including single and multi-agent settings. The potential impact of these contributions is significant, as enhancing the scalability of SPI algorithms opens up new possibilities for more robust and scalable applications of SPI in real-world environments. By addressing the computational limitations of state-of-the-art SPI algorithms, and SPIBB methods specifically, this work broadens the scope of problems that can be safely tackled with RL, contributing to the development of more reliable AI systems.

Future work will focus on several key areas: i) addressing bottlenecks related to the space complexity of SPI algorithms in large-scale problems; ii) exploring the theoretical foundations of these methods to establish stronger guarantees and better understand their limitations; iii) investigating the integration of the SPIBB criterion with other advanced techniques to enhance their applicability and performance; and iv) extending the SPIBB framework to continuous domains, which would require a substantial reformulation of the current theoretical foundations. This includes replacing count-based thresholds with function approximation and designing new safety constraints appropriate for continuous spaces. By pursuing these future directions, we aim to continue improving the practicality and robustness of safe policy improvement techniques, ultimately contributing to safer and more efficient decision-making processes in AI systems.

Acknowledgments

This paper has been prepared as a part of a collaboration between the University of Verona and Leonardo Labs belonging to Leonardo SpA. We also acknowledge financial support from PNRR MUR project PE0000013-FAIR and PR Veneto FESR 2021-2027 24729-002238 - SINERGHY.

References

P. Auer, N. Cesa-Bianchi, and P. Fischer. 2002. “Finite-Time Analysis of the Multiarmed Bandit Problem.” *Machine Learning*, 47, 2–3, 235–256.

R. Bellman. 1957. “A Markovian decision process.” *Journal of Mathematics and Mechanics*, 6, 5, 679–684.

D. P. Bertsekas and J. N. Tsitsiklis. 1996. *Neuro-dynamic programming.* Optimization and neural computation series. Vol. 3. Athena Scientific, I–XIII, 1–491.

F. Bianchi, A. Castellini, and A. Farinelli. 2025. “Scalable Safe Policy Improvement for Single and Multi-Agent Systems.” In: *Proceedings of the 11th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2024), AI*IA 2024* (CEUR Workshop Proceedings). Vol. 3956. CEUR-WS.org, 15–17.

L. Bonanni, D. Meli, A. Castellini, and A. Farinelli. 2025. “Monte Carlo Tree Search with Velocity Obstacles for safe and efficient motion planning in dynamic environments.” In: *Proceedings of the 2025 International Conference on Autonomous Agents and Multiagent Systems* (AAMAS ’25). IFAAMAS, Detroit, USA, 371–380.

S. Boucheron, G. Lugosi, and P. Massart. 2013. *Concentration Inequalities - A Nonasymptotic Theory of Independence.* Oxford University Press.

R. I. Brafman and M. Tennenholtz. 2003. “R-Max - a General Polynomial Time Algorithm for near-Optimal Reinforcement Learning.” *Journal of Machine Learning Research*, 3, 213–231.

C. Browne et al.. 2012. “A Survey of Monte Carlo Tree Search Methods.” *IEEE Transactions on Computational Intelligence and AI in Games*, 4, 1, 1–43.

A. Castellini, F. Bianchi, E. Zorzi, T. D. Simão, A. Farinelli, and M. T. J. Spaan. 2023. “Scalable Safe Policy Improvement via Monte Carlo Tree Search.” In: *Proceedings of the 40th International Conference on Machine Learning (ICML).* PMLR, 3732–3756.

A. Castellini, G. Chalkiadakis, and A. Farinelli. 2019. “Influence of state-variable constraints on Partially Observable Monte Carlo planning.” In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI).* ijcai.org, 5540–5546.

G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. 2008. “Monte Carlo Tree Search: A New Framework for Game AI.” In: *Proceedings of the 4th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* ((AIIDE)). AAAI Press, Stanford, California, 216–217.

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. 2021. "Decision Transformer: Reinforcement Learning via Sequence Modeling." In: *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., 15084–15097.

S. Choudhury, J. K. Gupta, P. Morales, and M. J. Kochenderfer. 2021. "Scalable Anytime Planning for Multi-Agent MDPs." In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)* (AAMAS 2021). IFAAMAS, 341–349.

R. Coulom. 2007. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." In: *Computers and Games*. Springer Berlin Heidelberg, 72–83.

E. Delage and S. Mannor. 2010. "Percentile Optimization for Markov Decision Processes with Parameter Uncertainty." *Operations Research*, 58, 1, 203–213.

E. Delage and S. Mannor. 2007. "Percentile Optimization in Uncertain Markov Decision Processes with Application to Efficient Exploration." In: *Proceedings of the 24th International Conference on Machine Learning* (ICML '07). Corvalis, Oregon, USA, 225–232.

G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. 2021. "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis." *Machine Learning*, 110, 9, 2419–2468.

D. Ernst, P. Geurts, and L. Wehenkel. 2005. "Tree-Based Batch Mode reinforcement learning." *Journal of Machine Learning Research*, 6, 503–556.

S. Fujimoto, D. Meger, and D. Precup. 2019. "Off-Policy Deep Reinforcement Learning without Exploration." In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2052–2062.

J. García and F. Fernández. 2015. "A Comprehensive Survey on Safe Reinforcement Learning." *Journal of Machine Learning Research*, 16, 42, 1437–1480.

C. Guestrin, D. Koller, R. E. Parr, and S. Venkataraman. 2003. "Efficient Solution Algorithms for factored MDPs." *Journal of Artificial Intelligence Research*, 19, 399–468.

R. A. Howard. 1960. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.

S. Katt, F. A. Oliehoek, and C. Amato. 2019. "Bayesian Reinforcement Learning in Factored POMDPs." In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. IFAAMAS, 7–15.

S. Katt, F. A. Oliehoek, and C. Amato. 2017. "Learning in POMDPs with Monte Carlo Tree Search." In: *Proceeding of the 34th International Conference on Machine Learning (ICML)*. PMLR, 1819–1827.

M. Kearns, Y. Mansour, and A. Y. Ng. 2002. "A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes." In: *Machine Learning*.

R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. 2020. "MOReL: Model-Based Offline Reinforcement Learning." In: *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 21810–21823.

M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray. 2022. *Algorithms for Decision Making*. MIT Press.

L. Kocsis, C. Szepesvari, and J. Willemson. 2006. "Improved Monte-Carlo Search." *University of Tartu, Estonia, Technical Report*, 1.

L. Kocsis and C. Szepesvári. 2006. "Bandit Based Monte-Carlo Planning." In: *Machine Learning: ECML 2006. 17th European Conference on Machine Learning* (LNCS). Vol. 4212. Springer-Verlag, 282–293.

A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. 2019. "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction." In: *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*. Curran Ass. Inc., 11761–11771.

A. Kumar, A. Zhou, G. Tucker, and S. Levine. 2020. "Conservative Q-Learning for Offline Reinforcement Learning." In: *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 1179–1191.

S. Lange, T. Gabel, and M. Riedmiller. 2012. "Batch reinforcement learning." In: *Reinforcement Learning: State-of-the-Art*. Springer Berlin Heidelberg, Berlin, Heidelberg, 45–73.

R. Laroche, P. Trichelair, and R. Tachet Des Combes. 2019. "Safe Policy Improvement with Baseline Bootstrapping." In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 3652–3661.

S. Levine, A. Kumar, G. Tucker, and J. Fu. 2020. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems." *CoRR*, abs/2005.01643. arXiv: 2005.01643.

G. Mazzi, A. Castellini, and A. Farinelli. 2021a. "Identification of Unexpected Decisions in Partially Observable Monte-Carlo Planning: A Rule-Based Approach." In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. IFAAMAS, 889–897.

G. Mazzi, A. Castellini, and A. Farinelli. 2023. "Risk-aware shielding of Partially Observable Monte Carlo Planning policies." *Artificial Intelligence*, 324, 103987.

G. Mazzi, A. Castellini, and A. Farinelli. 2021b. "Rule-based Shielding for Partially Observable Monte-Carlo Planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 243–251.

G. Mazzi, D. Meli, A. Castellini, and A. Farinelli. 2023. "Learning Logic Specifications for Soft Policy Guidance in POMCP." In: *Proceedings of the 22nd International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. IFAAMAS, 373–381.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. *Playing atari with deep reinforcement learning*. (2013).

K. Nadjahi, R. Laroche, and R. Tachet des Combes. 2019. "Safe policy improvement with soft baseline bootstrapping." In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 53–68.

C. Papadimitriou and J. Tsitsiklis. Aug. 1987. "The Complexity of Markov Decision Processes." *Math. Oper. Res.*, 12, 3, (Aug. 1987), 441–450. doi:10.1287/moor.12.3.441.

M. Petrik, M. Ghavamzadeh, and Y. Chow. 2016. "Safe policy improvement by minimizing robust baseline regret." In: *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS)*. Curran Associates Inc., Barcelona, Spain, 2306–2314.

R. F. Prudencio, M. R. O. A. Maximo, and E. L. Colombini. 2022. *A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems*. (2022).

M. L. Puterman. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

"Uncertainty in Reinforcement Learning - Awareness, Quantisation, and Control." *Robot Learning*. InTech, 65–90.

D. Schneegass, S. Udluft, and T. Martinetz. 2008. "Uncertainty propagation for quality assurance in Reinforcement Learning." In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2588–2595.

P. Scholl, F. Dietrich, C. Otte, and S. Udluft. 2022a. "Safe Policy Improvement Approaches and Their Limitations." In: *Agents and Artificial Intelligence*. Springer International Publishing, Cham, 74–98.

P. Scholl, F. Dietrich, C. Otte, and S. Udluft. 2022b. "Safe Policy Improvement Approaches on Discrete Markov Decision Processes." *CoRR*, abs/2201.12175. arXiv: 2201.12175.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. *Proximal policy optimization algorithms*. (2017).

D. Silver, A. Huang, et al.. 2016. "Mastering the game of Go with deep neural networks and tree search." *Nature*, 529, 7587, 484–489.

D. Silver and J. Veness. 2010. "Monte-Carlo planning in large POMDPs." In: *Proceedings of the 23rd Conference on Neural Information Processing Systems (NeurIPS)*. Curran Ass., Vancouver, British Columbia, Canada, 2164–2172.

T. D. Simão and M. T. J. Spaan. 2019. "Safe policy improvement with baseline bootstrapping in factored environments." In: *the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 4967–4974.

R. Sutton and A. Barto. 2018. *Reinforcement Learning, An Introduction*. (2nd ed.). MIT Press.

P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. 2015. "High confidence policy improvement." In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*. PMLR, 2380–2388.

O. Vinyals et al.. 2019. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." *Nature*, 575, 7782, 350–354.

E. Zorzi, A. Castellini, L. Bakopoulos, G. Chalkiadakis, and A. Farinelli. 2025. "Seldonian Reinforcement Learning for Ad Hoc Teamwork." *Reinforcement Learning Journal*, 2, 1–19.

A Theoretical Analysis: Full Version

In this appendix, we provide the full version of the theoretical analysis, with complete proofs.

A.1 Convergence of MCTS-SPIBB to SPIBB

The convergence of UCT to an optimal policy given an unconstrained policy space is here implemented in the context of safe policy improvement with baseline bootstrapping, in which the policy has to satisfy the *SPIBB constraint*, namely, the improved policy must belong to the subspace $\Pi_0 = \{\pi \in S \rightarrow \mathcal{P}(A) | \pi(s,a) = \pi_0(s,a) : \forall(s,a) \in \mathcal{B}\}$, with $\mathcal{B}$ set of state-action pairs that occur less than $N_\wedge$ times in dataset $\mathcal{D}$. This constraint requires that for each state encountered in simulations, we consider in different ways bootstrapped and non-bootstrapped actions. In particular, non-bootstrapped actions can still be selected using the UCT strategy, which is proven to converge to optimality, while bootstrapped actions (i.e., actions that have not been observed enough times in dataset $\mathcal{D}$) must be selected according to the related baseline probabilities $\pi_0(s,a)$. The bias of the estimated state-value (i.e., expected average payoff) of the root node of a MCTS computed by UCT tends to zero as the number of simulations tends to infinity (Kocsis, Szepesvari, et al. 2006). In other words, the estimated state value of the root node tends to be its optimal value, and the action with maximum Q-value tends to be the optimal one as the number of simulations tends to infinity. Our proof adds to the bias of the estimated state-value (related to UCT for non-bootstrapped actions) the bias due to baseline policy sampling (for bootstrapped actions) and shows that the composed bias still converges to zero as the number of simulations tends to infinity. We notice that the two contributions mix in the lower levels of the MC tree, hence the values of all (bootstrapped and non-bootstrapped) actions of the root node are affected by contributions of bootstrapped and non-bootstrapped actions of the underlying nodes. This makes the analysis non-trivial and of interest.

Table 1. Summary of mathematical notation.

Symbol	Description
$\mathcal{B}_A$	Set of bootstrapped actions for the current state
$\overline{\mathcal{B}}_A$	Set of non-bootstrapped actions for the current state
L	Number of bootstrapped actions for the current state
K	Number of non-bootstrapped actions for the current state
$X_{i,t}$	Payoff (i.e., return) obtained by selecting action i in the t-th simulation
C_p	Constant regulating exploration-exploitation tradeoff in UCT
m	Number of simulations performed from the current state
c	Number of bootstrapped actions selected from the current state
n	Number of non-bootstrapped actions selected from the current state
$m = c + n$	Relationship between m, c and n
$T_i(m) \coloneqq \sum_{t\in\{1,\dots,m\}} \mathbb{1}[I_t = i]$	Number of times action i was selected in current state after m simulations
$\bar{X}_{i,m} \coloneqq \frac{1}{m}\sum_{t=1}^{m} X_{i,t}$	Average payoff of action i after m simulations from the current state
$\mu_{i,m} \coloneqq \mathbb{E}\{\bar{X}_{i,m}\}$	Expected value of the average payoff of action i after m simulations
$\mu_i \coloneqq \lim_{m\to\infty} \mu_{i,m}$	Expected value of the average payoff of action i in the the limit
$\lvert\delta_{i,m}\rvert \coloneqq \lvert\mu_{i,m} - \mu_i\rvert$	Bias of estimated expected average payoff at simulation m
$\mu_n^\star, \mu^\star, X_t^\star, \bar{X}_n^\star$	Symbols defined above but related to the optimal non-bootstrapped action
$\Delta_i \coloneqq \mu^\star - \mu_i$	Diff. btw exp. val. of avg payoff optimal and another non-bootstrapped act.
$N_0(\epsilon)$	if $t \geq N_0(\epsilon)$ then for non-bootstrapped actions $\lvert\delta_{i,t}\rvert \leq \epsilon\Delta_i/2$ and $\lvert\delta_{j^*,t}\rvert \leq \epsilon\Delta_i/2$
$p_{\mathcal{B}} \coloneqq \sum_{i\in\mathcal{B}_A} \pi_0(i)$	Probability of selecting a bootstrapped action from the current state
$p_{\overline{\mathcal{B}}} \coloneqq 1 - p_{\mathcal{B}}$	Probability to select a non-bootstrapped action from the current state
$\lim_{m\to\infty} \frac{c(m)}{m} = p_{\mathcal{B}}$	Value of c in the limit
$\lim_{m\to\infty} \frac{n(m)}{m} = p_{\overline{\mathcal{B}}}$	Value of n in the limit
$\lim_{m\to\infty} \frac{T_i(m)}{m} = \pi_0(i), \forall i \in \mathcal{B}_A$	Value of $\frac{T_i(m)}{m}$ in the limit
$\lim_{m\to\infty} \frac{T_i(c(m))}{m} = \pi_0(i), \forall i \in \mathcal{B}_A$	Value of $\frac{T_i(c(m))}{m}$ in the limit
$\lim_{c\to\infty} \frac{T_i(c)}{c} = \frac{\pi_0(i)}{p_{\mathcal{B}}}, \forall i \in \mathcal{B}_A$	Value of $\frac{T_i(c)}{c}$ in the limit

A.1.1 Notation. We indicate actions in the set $A = \{1, \dots, |A|\}$ as $i \in A$ (considering each action as its index in A), bootstrapped actions for the current state as $i \in \mathcal{B}_A$ (for the sake of compactness, we omit from $\mathcal{B}_A(s)$ symbol s for the state when we refer to the current state), and non-bootstrapped actions for the current state as $i \in \overline{\mathcal{B}}_A$. We use L to denote the number of bootstrapped actions for the current state and K to denote the number of non-bootstrapped actions. Being i an action index and t the current simulation index, we indicate with $X_{i,t}$ the (random) payoff (i.e., return) obtained by selecting i in the t-th simulation. Formally, this payoff is a realization of a random variable with a non-stationary distribution (i.e., the distribution can change across epochs t). Given m total simulations, the number of times we have selected action i up to that point is $T_i(m) \coloneqq \sum_{t\in\{1,\dots,m\}} \mathbb{1}[I_t = i]$, where I_t is a random variable representing the action taken in the t-th simulation. The average payoff after m simulations is indicated as $\bar{X}_{i,m} \coloneqq \frac{1}{m}\sum_{t=1}^{m} X_{i,t}$.

As mentioned above, payoff sequences are non-stationary, namely, taking the same action multiple times can result in payoffs taken from different distributions. Formally, if we fix an action $i \in A$, and a payoff sequence $X_{i,1}, X_{i,2}, \ldots, X_{i,m}$, then $X_{i,t} \sim \mathcal{P}_t$ and $X_{i,t'} \sim \mathcal{P}_{t'}$ with $\mathcal{P}_t$ possibly different from $\mathcal{P}_{t'}$. We indicate with $\mu_{i,m} \coloneqq \mathbb{E}\{\bar{X}_{i,m}\}$ the expected value of the average payoff of action i after m simulations, and with μ_i its value in the limit $\mu_i \coloneqq \lim_{m\to\infty} \mu_{i,m}$ (see Assumption A.5 for its existence). We also define $\delta_{i,m} \coloneqq \mu_{i,m} - \mu_i$ as the difference, at simulation m, between the expectation of the current mean and its value in the limit. For the non-bootstrapped actions, we assume there exists only a single optimal action $i^\star$, as in (Kocsis, Szepesvari, et al. 2006)[9]. For quantities related to this action we drop the i symbol and use the $\star$ symbol, e.g., $\mu^\star_m, \mu^\star, X^\star_t, \bar{X}^\star_m$. We define the difference between the expected value of the average payoff of the optimal action μ^* and the expected value of the average payoff of another action i, namely μ_i, as $\Delta_i \coloneqq \mu^\star - \mu_i$. This value is always positive. Since $\delta_{i,t}$ converges by assumption to zero, for all $\epsilon > 0$ there exists and index $N_0(\epsilon)$ such that if $t \geq N_0(\epsilon)$ then $|\delta_{i,t}| \leq \epsilon\Delta_i/2$ and $|\delta_{j^*,t}| \leq \epsilon\Delta_i/2$, where i is a sub-optimal action and j^* is the optimal action. This says that if enough (i.e., $\geq N_0(\epsilon)$) simulations are run then the estimated expected average payoff of the optimal action will be far enough from the estimated expected average payoffs of all sub-optimal actions, hence, in that case, the optimal action can be identified as the action with the highest estimated expected average payoff. In the following, we will arbitrarily use $\epsilon = 1/2$, as in (Kocsis, Szepesvari, et al. 2006), for which it holds that if $t \geq N_0(1/2)$ then $|\delta_{i,t}| \leq \Delta_i/4$ and $|\delta_{j^*,t}| \leq \Delta_i/4$, therefore each estimated expected average payoff of sub-optimal action is farther than $\Delta_i/2$ from the estimated expected average payoff of the optimal action.

We then define the probability of selecting a bootstrapped action from the current state as $p_{\mathcal{B}} \coloneqq \sum_{i\in\mathcal{B}_A} \pi_0(i)$ and the probability of selecting a non-bootstrapped action as $p_{\overline{\mathcal{B}}} \coloneqq 1 - p_{\mathcal{B}}$. Also in this case we omit the symbol s since we refer to the current state. Over m simulations performed from the current state, we assume c simulations selected a bootstrapped action and n simulations selected a non-bootstrapped action, hence $m = c + n$. For m tending to ∞ we have that c/m tends to $p_{\mathcal{B}}$, hence we write the limit $\lim_{m\to\infty} \frac{c(m)}{m} = p_{\mathcal{B}}$ (symbol $c(m)$ is used only here to emphasize that c depends on m). Similarly, with non-bootstrapped actions, for m tending to ∞ we have that n/m tends to $p_{\overline{\mathcal{B}}}$, hence we write the limit $\lim_{m\to\infty} \frac{n(m)}{m} = p_{\overline{\mathcal{B}}}$. We then represent by $T_i(c)$ the number of times bootstrapped action $i \in \mathcal{B}_A$ has been selected over c selections of bootstrapped actions and by $T_i(n)$ the number of times non-bootstrapped action $i \in \overline{\mathcal{B}}_A$ has been selected over n selections of non-bootstrapped actions. Focusing on single bootstrapped actions we have that $\forall i \in \mathcal{B}_A, T_i(m)/m$ tends to $\pi_0(i)$ as m tends to ∞. Therefore we can also write the limit $\lim_{m\to\infty} \frac{T_i(m)}{m} = \pi_0(i), \forall i \in \mathcal{B}_A$. The limit can also be written as $\lim_{m\to\infty} \frac{T_i(c(m))}{m} = \pi_0(i), \forall i \in \mathcal{B}_A$ since $T_i(m) = T_i(c(m))$ (the first notation refers to the number of times action i is selected over m selections of bootstrapped or non-bootstrapped actions, while the second notation refers to the number of times action i is selected over $c(m)$ selections of bootstrapped actions only, but the number is the same). Finally, the same limit can also be written in terms of c in the following form $\lim_{c\to\infty} \frac{T_i(c)}{c} = \frac{\pi_0(i)}{p_{\mathcal{B}}}, \forall i \in \mathcal{B}_A$. Notice that the ratio $\frac{\pi_0(i)}{p_{\mathcal{B}}}$ is used in line 5 of Algorithm 6 for selecting bootstrapped actions. Table 1 summarizes the mathematical notation introduced so far and used in the theoretical analysis.

A.1.2 Derivation. The average payoff of the root state of a MC tree considering only non-bootstrapped actions is (Kocsis, Szepesvari, et al. 2006)

$$\bar{X}_n \coloneqq \frac{1}{n}\sum_{i\in\overline{\mathcal{B}}_A} T_i(n)\bar{X}_{i,T_i(n)}. \tag{19}$$

[9]Notice that if all actions are bootstrapped then $n = 0$, $c = m$ and $\bar{X}_m = \bar{X}_c$. On the other hand, if all actions are non-bootstrapped then $c = 0$, $n = m$, and $\bar{X}_m = \bar{X}_n$.

This value is proved to converge to the value of the optimal policy when UCT is used as action selection strategy. We define in a similar way the average payoff of the bootstrapped actions, namely,

$$\bar{X}_c \coloneqq \frac{1}{c} \sum_{i \in \mathcal{B}_A} T_i(c) \bar{X}_{i,T_i(c)}. \tag{20}$$

We have therefore split in two parts the average payoff of the root node of our MC tree, corresponding to the current state of the agent. Putting together in a weighted way the two terms, we obtain the total average payoff

$$\begin{aligned} \bar{X}_m &\coloneqq \frac{1}{m}(c \cdot \bar{X}_c + n \cdot \bar{X}_n) \\ &= \frac{c}{m} \cdot \bar{X}_c + \frac{n}{m} \cdot \bar{X}_n \\ &= \frac{1}{m} \sum_{i \in \mathcal{B}_A} T_i(c) \bar{X}_{i,T_i(c)} + \frac{1}{m} \sum_{i \in \overline{\mathcal{B}}_A} T_i(n) \bar{X}_{i,T_i(n)} \\ &= \frac{1}{m} (\sum_{i \in \mathcal{B}_A} T_i(c) \bar{X}_{i,T_i(c)} + \sum_{i \in \overline{\mathcal{B}}_A} T_i(n) \bar{X}_{i,T_i(n)}) \end{aligned} \tag{21}$$

Our proof aims to show that the difference between the estimated expected average payoff $\mathbb{E}\{\bar{X}_m\}$ (i.e., the expected value of the current state) computed using MCTS-SPIBB and the true expected average payoff of the optimal policy (i.e., the true optimal expected value of the current state) in Π_0 (i.e., the space of policies satisfying the SPIBB constraint) tends to zero as the number of simulations m tends to infinity. In the following, we call this difference *bias* of the estimated expected average payoff of the current state or *bias* of the expected value of the current state. In this derivation, we prove that UCT and the baseline policy can be used together to select optimal actions in the simulations, achieving a MCTS-based version of SPIBB that computes optimal policies in Π_0.

We will make use of the following standard concentration inequalities:

Fact A.1 (Hoeffding's inequality (Boucheron et al. 2013))**.** Let $Y_1, \ldots, Y_n$ be independent random variables such that $a_p \le Y_p \le b_p$ almost surely for all $p < n$. Let $S_n = \sum_{p=1}^{n} (Y_p - \mathbb{E}(Y_p))$. Then for every $\lambda > 0$

$$\mathbb{P}\{S_n \ge \lambda\} \le exp\left(-\frac{2\lambda^2}{\sum_{p=1}^{n} (b_p - a_p)^2}\right) \tag{22}$$

Fact A.2 (Hoeffding-Azuma inequality (Kocsis, Szepesvari, et al. 2006))**.** Let $Y_1, \ldots, Y_n$ be a martingale difference (i.e., $\mathbb{E}(Y_p | Y_1, \ldots, Y_{p-1}) = Y_{p-1}$) with $|Y_p| \le C$ and $C > 0$. Let $S_n = \sum_{p=1}^{n} Y_p$, then for every $\epsilon > 0$

$$\mathbb{P}\{S_n \ge \epsilon n\} \le \exp\left(-\frac{2n\epsilon^2}{C^2}\right) \tag{23}$$

$$\mathbb{P}\{S_n \le -\epsilon n\} \le \exp\left(-\frac{2n\epsilon^2}{C^2}\right) \tag{24}$$

Then, from (Kocsis, Szepesvari, et al. 2006) we get the following concentration inequality for $\bar{X}_{i,T_i(t)} - \mu_i$ on non-bootstrapped actions.

Fact A.3 (Inequality for bias $\bar{X}_{i,T_i(t)} - \mu_i$ on non-bootstrapped actions (Kocsis, Szepesvari, et al. 2006))**.** Let X_{it} be payoffs i.i.d. (or a form of martingale difference process shifted by a constant) and $c_{t,T_i(t)} = \sqrt{\frac{2 \ln t}{T_i(t)}}$ the bias sequence of UCT, then

$$\mathbb{P}\left\{\bar{X}_{i,T_i(t)} \geq \mu_i + c_{t,T_i(t)}\right\} \leq t^{-4} \tag{25}$$

$$\mathbb{P}\left\{\bar{X}_{i,T_i(t)} \leq \mu_i - c_{t,T_i(t)}\right\} \leq t^{-4} \tag{26}$$

This inequality follows from Hoeffding-Azuma inequality (Fact A.2), considering $Y_p = X_{i,p} - \mu_i$ with $p = 1, \ldots, T_i(t)$, which is a martingale difference with $|Y_p| \leq 1$, since $|X_{i,p} - \mu_i| \leq 1$ with $p > 0$. Hence, we use $C = 1$ and $S_{T_i(t)} = \sum_{p=1}^{T_i(t)} Y_p = T_i(t)(\bar{X}_{i,T_i(t)} - \mu_i)$, with $\epsilon = c_{t,T_i(t)} = \sqrt{\frac{2 \ln t}{T_i(t)}}$. Substituting these values in Equation 23 we obtain

$$\mathbb{P}\left\{T_i(t)(\bar{X}_{i,T_i(t)} - \mu_i) \geq T_i(t)\sqrt{\frac{2 \ln t}{T_i(t)}}\right\} \leq \exp\left(-\frac{2T_i(t)\frac{2 \ln t}{T_i(t)}}{1}\right)$$

from which is simply derived Equation 25. Equation 26 is obtained similarly from Equation 24.

We then extend Fact A.3 to bootstrapped actions, obtaining the following concentration inequality for the payoff bias sequence generated by bootstrapped actions.

Fact A.4 (Inequality for bias $\bar{X}_{i,T_i(t)} - \mu_i$ on bootstrapped actions)**.** Let X_{it} be payoffs i.i.d. (or a form of martingale difference process shifted by a constant) and $\pi_0(i), i \in \mathcal{B}_A$, the baseline probabilities for bootstrapped actions from the current state, then

$$\mathbb{P}\left\{\bar{X}_{i,T_i(t)} \geq \mu_i + \sqrt{\frac{2 \ln t}{T_i(t)}}\right\} \leq t^{-4\pi_0(i)} \tag{27}$$

$$\mathbb{P}\left\{\bar{X}_{i,T_i(t)} \leq \mu_i - \sqrt{\frac{2 \ln t}{T_i(t)}}\right\} \leq t^{-4\pi_0(i)} \tag{28}$$

Let us start the actual derivation by defining an assumption that characterizes the payoff sequences used in the rest of the proof.

Assumption A.5. Let I_t be a discrete action index set, namely a random variable representing the action taken at the t-th action selection from the current state. In MCTS-SPIBB I is computed in two steps (see Figure 1), first selecting between bootstrapped and non-bootstrapped actions according to probability $p_{\mathcal{B}}$ (see line 2 of Algorithm 6), then, in case of non-bootstrapped action, using the UCT strategy, i.e., $I_t = \operatorname{argmax}_{i \in \overline{\mathcal{B}}_A(s)} \{\bar{X}_{i,T_i(t-1)} + 2C_p\sqrt{\frac{\ln(t-1)}{T_i(t-1)}}\}$ (see (Kocsis, Szepesvari, et al. 2006) and Algorithm 6, line 11, in this paper), and in case of bootstrapped action according to the probabilistic strategy $I \sim \pi_0(s,.)/p_{\mathcal{B}}$ (see Algorithm 6, line 3-7, in this paper), hence each bootstrapped action i is selected with probability $\pi_0(s,i)/p_{\mathcal{B}}$. Index set I_t defines a filtration $\{\mathcal{F}_{i,t}\}_t$ such that $\{X_{i,t}\}_t$ is $\mathcal{F}_{i,t}$-adapted and $X_{i,t}$ is conditionally independent of $\mathcal{F}_{i,t+1}, \mathcal{F}_{i,t+2}, \ldots$ given $\mathcal{F}_{i,t-1}$ (Kocsis, Szepesvari, et al. 2006). Then we assume $0 \leq X_{i,t} \leq 1$ and that the limit of $\mu_{i,m} = \mathbb{E}\{\bar{X}_{i,m}\}$ exists both for non-bootstrapped and bootstrapped actions. Furthermore, we assume there exist a constant $C_p > 0$ and an integer N_p such that for $n > N_p$ and for any $\delta > 0$, with $\Delta_n(\delta) \coloneqq C_p\sqrt{n \ln(1/\delta)}$, the following concentration bounds hold for all actions $i \in A$:

$$\mathbb{P}\left\{n\bar{X}_{i,n} \geq n\mathbb{E}\left\{\bar{X}_{i,n}\right\} + \Delta_n(\delta)\right\} \leq \delta \tag{29}$$

$$\mathbb{P}\left\{n\bar{X}_{i,n} \leq n\mathbb{E}\left\{\bar{X}_{i,n}\right\} - \Delta_n(\delta)\right\} \leq \delta \tag{30}$$

Concentration bounds of Equations 29 and 30 hold due to Hoeffding inequality (see Fact A.1), considering the payoffs $X_{i,t}$ as independent random variables. Their values are $0 \leq X_{i,t} \leq 1$, hence $a_t = 0$ and $b_t = 1$ for

$1 \leq t \leq T_i(t)$. Then we define $S_{T_i(t)} = \sum_{t=1}^{T_i(t)} (X_{i,t} - \mathbb{E}\left\{X_{i,t}\right\})$ and $\lambda = \Delta_{T_i(t)}(\delta) = C_p\sqrt{T_i(t)\ln(1/\delta)}$. Substituting these values in Equation 22 we get

$$\mathbb{P}\left\{\sum_{t=1}^{T_i(t)} (X_{i,t} - \mathbb{E}\left\{X_{i,t}\right\}) \geq \Delta_{T_i(t)}(\delta)\right\} \leq \exp\left(-\frac{2C_p^2 T_i(t)\ln(1/\delta)}{T_i(t)}\right)$$

from which Equations 29 and 30 are simply derived with $C_p = \frac{1}{\sqrt{2}}$. As a first result we provide a bound on the difference between the expected average payoff $\mathbb{E}\{\bar{X}_m\}$ estimated by MCTS-SPIBB after m simulations and the optimal average payoff satisfying the SPIBB constraint, namely, $\sum_{i\in\mathcal{B}_A} \pi_0(i)\mu_i + p_{\overline{\mathcal{B}}} \cdot \mu^\star$. This is, in practice, the bias of the value of the current state (i.e., root node) with respect to the optimal value in Π_0. The action selection strategy used in the following Theorem A.6 mixes UCT, for non-bootstrapped actions, and baseline policy, for bootstrapped actions. The mix is performed in a non-stationary context in which payoffs can drift since several actions must be selected in sequence, considering also states reached in successive time instants. Notice that also the payoff of bootstrapped actions can drift because of the influence of UCT in the low levels of the sub-trees used for the estimation of these payoffs.

Theorem A.6 (Bias of the mean.). *Let*

$$\bar{X}_m = \frac{1}{m}\sum_{i\in\mathcal{B}_A} T_i(c)\bar{X}_{i,T_i(c)} + \frac{1}{m}\sum_{i\in\overline{\mathcal{B}}_A} T_i(n)\bar{X}_{i,T_i(n)} \tag{31}$$

Under Assumption A.5 the following bound holds

$$\left|\mathbb{E}\{\bar{X}_m\} - \sum_{i\in\mathcal{B}_A} \pi_0(i)\cdot\mu_i - p_{\overline{\mathcal{B}}}\cdot\mu^\star\right| \leq \sum_{i\in\mathcal{B}_A} \pi_0(i)\cdot|\delta_{i,T_i(c)}| + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| + O\left(\frac{K(C_p^2\ln m + N_0)}{m}\right) \tag{32}$$

where $N_0 = N_0(\epsilon)$.

Proof. First, we notice that the term $\sum_{i\in\mathcal{B}_A} \pi_0(i)\mu_i$ in Equation 32 represents the true optimum value of the weighted sum of the average payoffs obtained from bootstrapped actions, i.e., it is the equivalent (for bootstrapped actions) of value $p_{\overline{\mathcal{B}}}\cdot\mu^\star$ for the optimal non-bootstrapped action. Let $\bar{X}_c \coloneqq \frac{1}{c}\sum_{i\in\mathcal{B}_A} T_i(c)\bar{X}_{i,T_i(c)}$ (see Equation 20), $\bar{X}_n \coloneqq \frac{1}{n}\sum_{i\in\overline{\mathcal{B}}_A} T_i(n)\bar{X}_{i,T_i(n)}$ (see Equation 19) and $\bar{X}_m \coloneqq \frac{c}{m}\cdot\bar{X}_c + \frac{n}{m}\cdot\bar{X}_n$ (see Equation 21). For the linearity of the expectation, we can rewrite the expected payoff $\mathbb{E}\{\bar{X}_m\}$ as follows

$$\mathbb{E}\{\bar{X}_m\} = \mathbb{E}\left\{\frac{c}{m}\cdot\bar{X}_c + \frac{n}{m}\cdot\bar{X}_n\right\} = \frac{c}{m}\mathbb{E}\{\bar{X}_c\} + \frac{n}{m}\mathbb{E}\{\bar{X}_n\} \tag{33}$$

Therefore, the left part of Inequality 32 can be written as

$$
\begin{aligned}
\left|\mathbb{E}\{\bar{X}_m\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i - p_{\overline{\mathcal{B}}}\cdot\mu^\star\right| &= \left|\left(\frac{c}{m}\mathbb{E}\{\bar{X}_c\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i\right)\right. \\
&\quad \left. + \left(\frac{n}{m}\mathbb{E}\{\bar{X}_n\} - p_{\overline{\mathcal{B}}}\cdot\mu^\star\right)\right| && (34) \\
\left[\textit{Since } \lim_{m\to\infty}\tfrac{c(m)}{m} = p_{\mathcal{B}} \;\; \textit{and} \;\; \lim_{m\to\infty}\tfrac{n(m)}{m} = p_{\overline{\mathcal{B}}}\right] \quad &= \left|(p_{\mathcal{B}}\mathbb{E}\{\bar{X}_c\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i)\right. \\
&\quad \left. + (p_{\overline{\mathcal{B}}}\mathbb{E}\{\bar{X}_n\} - p_{\overline{\mathcal{B}}}\cdot\mu^\star)\right| && (35) \\
&= \left|(p_{\mathcal{B}}\mathbb{E}\{\bar{X}_c\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i)\right. \\
&\quad \left. + p_{\overline{\mathcal{B}}}(\mathbb{E}\{\bar{X}_n\} - \mu^\star)\right| && (36) \\
[\textit{Triangle inequality}] \quad &\le \left|p_{\mathcal{B}}\mathbb{E}\{\bar{X}_c\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i\right| \\
&\quad + p_{\overline{\mathcal{B}}}\left|\mathbb{E}\{\bar{X}_n\} - \mu^\star\right| && (37) \\
[\textit{Theorem 3 of (Kocsis, Szepesvari, et al. 2006)}] \quad &\le \left|p_{\mathcal{B}}\mathbb{E}\{\bar{X}_c\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i\right| \\
&\quad + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \\
&\quad + O\left(p_{\overline{\mathcal{B}}}\frac{K(C_p^2\ln n + N_0)}{n}\right) && (38) \\
\left[\mathbb{E}\{\bar{X}_c\} = \textstyle\sum_{i\in\mathcal{B}_A}\tfrac{T_i(c)}{c}\mathbb{E}\{\bar{X}_{i,T_i(c)}\}\right] \quad &= \left|p_{\mathcal{B}}\sum_{i\in\mathcal{B}_A}\frac{T_i(c)}{c}\mathbb{E}\{\bar{X}_{i,T_i(c)}\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i\right| \\
&\quad + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \\
&\quad + O\left(\frac{n}{m}\frac{K(C_p^2\ln n + N_0)}{n}\right) \\
\left[\lim_{c\to\infty}\tfrac{T_i(c)}{c} = \tfrac{\pi_0(i)}{p_{\mathcal{B}}}; \mu_{i,T_i(c)} \coloneqq \mathbb{E}\{\bar{X}_{i,T_i(c)}\}\right] \quad &= \left|\sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_{i,T_i(c)} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i\right| \\
&\quad + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \\
&\quad + O\left(\frac{K(C_p^2\ln n + N_0)}{m}\right) && (39) \\
\left[\lim_{m\to\infty}\tfrac{n(m)}{m} = p_{\overline{\mathcal{B}}}\right] \quad &= \left|\sum_{i\in\mathcal{B}_A}\pi_0(i)(\mu_{i,T_i(c)} - \mu_i)\right| \\
&\quad + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \\
&\quad + O\left(\frac{K(C_p^2\ln(p_{\overline{\mathcal{B}}}\cdot m) + N_0)}{m}\right) && (40)
\end{aligned}
$$

$$\begin{aligned}
&[\,\delta_{i,T_i(c)} \coloneqq \mu_{i,T_i(c)} - \mu_i;\ \textit{Prop. of logarithms}] &= \left|\sum_{i\in\mathcal{B}_A} \pi_0(i)\cdot\delta_{i,T_i(c)}\right| \\
&&+\, p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \\
&&+\, O\left(\frac{K(C_p^2\ln p_{\overline{\mathcal{B}}} + C_p^2\ln m + N_0)}{m}\right) \qquad (41)\\
&[\,C_p^2\ln p_{\overline{\mathcal{B}}} \le 0 \to \textit{removed}] &= \left|\sum_{i\in\mathcal{B}_A} \pi_0(i)\cdot\delta_{i,T_i(c)}\right| \\
&&+\, p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \\
&&+\, O\left(\frac{K(C_p^2\ln m + N_0)}{m}\right) \qquad (42)\\
&[\,\textit{Triangle inequality}] &\le \sum_{i\in\mathcal{B}_A} \pi_0(i)\cdot|\delta_{i,T_i(c)}| \\
&&+\, p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \\
&&+\, O\left(\frac{K(C_p^2\ln m + N_0)}{m}\right) \qquad (43)
\end{aligned}$$

□

The bound provided by Theorem A.6 depends on the biases $\delta_{i,T_i(c)}$ of the values of bootstrapped actions in the current state and the bias δ_n^* of the optimal non-bootstrapped action in the current state. Namely, it considers only (action-)nodes of the MC tree one level below the root. The next theorem extends this result, considering all the levels of the MC tree up to the leaves and providing a bound that depends only on the total number of simulations m, the depth of the tree D and the number of non-bootstrapped and bootstrapped actions, K and L, respectively. The bound converges to zero as m increases. This proves that the value generated by MCTS-SPIBB for the current state converges to the optimal value in Π_0 (as SPIBB).

Theorem A.7 (Convergence of the estimated expected payoff $\bar{X}_m$.). *Consider algorithm MCTS-SPIBB running on a tree of depth D, branching factor $|A| = |\mathcal{B}_A| + |\overline{\mathcal{B}}_A| = L + K$ with $0 \le L \le |A|$ bootstrapped actions and $0 \le K \le |A|$ non-bootstrapped actions in each state, and stochastic payoffs at the leaves. Assume that the payoffs lie in the interval* $[0,1]$. *Then the bias of the estimated expected payoff, $\bar{X}_m$, is*

$$O\left(\frac{L^D K D\ln m + L^D K^D}{m}\right). \qquad (44)$$

Proof. The proof is made by induction on D.
Base. Consider the case with $D = 1$. The bias of $\mathbb{E}\left\{\bar{X}_m\right\}$ can be bounded, according to Theorem A.6, as

$$\left|\mathbb{E}\{\bar{X}_m\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i - p_{\overline{\mathcal{B}}}\cdot\mu^\star\right| \le \sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}| + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| + O\left(\frac{K(\ln m + N_0)}{m}\right). \qquad (45)$$

With $D = 1$, the biases of bootstrapped actions $|\delta_{i,T_i(c)}|$ and the optimal non-bootstrapped action $|\delta_n^*|$ refer to leaf nodes and UCT on non-bootstrapped actions corresponds to UCB1. Since $0 \le X_{i,t} \le 1$ and μ_i exist $\forall i \in A$, for Assumption A.5, we have that $|\delta_{i,T_i(c)}| \coloneqq |\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} - \mu_i| \le 1, \forall i \in \mathcal{B}_A$ and $|\delta_n^*| \coloneqq |\mathbb{E}\left\{\bar{X}_{i^*,n}\right\} - \mu^*| \le 1$, therefore

$$\sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}| + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| \le p_{\mathcal{B}} + p_{\overline{\mathcal{B}}} = 1. \qquad (46)$$

Then, the term $O\left(\frac{K\ln m+KN_0}{m}\right)$ of Equation 45 is also an $O\left(\frac{LK\ln m+LK}{m}\right)$ (i.e., theorem thesis with $D=1$) because *i)* ($K\ln m \leq LK\ln m$) since $L \geq 1$ (otherwise we are in the case of only non-bootstrapped actions for which Th. 7 of (Kocsis, Szepesvari, et al. 2006) holds), *ii)* ($KN_0 \leq LK$) since $L \geq 1$ and $N_0 = O(L^D K^{D-1})$, that is, $N_0 = O(L)$ with $D=1$. Notice that with $D=1$ N_0 is a constant that depends neither on L nor on K.

Inductive step. Let us assume that the theorem holds for all trees of depth $D-1$, namely, that for each of these trees the bias of the estimated expected payoff is

$$O\left(\frac{L^{D-1}K(D-1)\ln m + L^{D-1}K^{D-1}}{m}\right). \tag{47}$$

then we show that also for all trees of depth D the theorem holds. Theorem A.6 allows us to connect the root node of a tree of depth D with its child nodes, which are root nodes of subtrees of depth $D-1$. According to that theorem, the bias of the estimated expected payoff in the root (depth D) is

$$\left|\mathbb{E}\{\bar{X}_m\} - \sum_{i\in\mathcal{B}_A}\pi_0(i)\mu_i - p_{\overline{\mathcal{B}}}\cdot\mu^\star\right| \leq \sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}| + p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| + O\left(\frac{K(\ln m + N_0)}{m}\right). \tag{48}$$

where $|\delta_{i,T_i(c)}| = |\mu_{i,T_i(c)} - \mu_i| = |\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} - \mu_i|$, $i \in \mathcal{B}_A$ are the biases of the estimated expected payoffs of nodes reachable performing a bootstrapped action from the root, and $|\delta_n^*| = |\mu_n^* - \mu^*| = |\mathbb{E}\left\{\bar{X}_{i^*,n}\right\} - \mu^*|$ is the bias of the estimated expected payoffs of the node reachable performing the optimal non-bootstrapped action (identified using the UCT strategy) from the root. Since the nodes reached performing a bootstrapped or the optimal non-bootstrapped action are roots of a sub-tree with depth reduced by 1 w.r.t. the root, biases $|\delta_{i,T_i(c)}|$ and $|\delta_n^*|$ refer to trees of depth $D-1$ for which the inductive hypothesis holds.

We consider now each of the three terms on the right of inequality (48) separately. Then, we put the results together to complete the proof. The first term is related to the sum of biases of bootstrapped actions, namely, $\sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}|$. We consider each action i separately and apply the inductive hypothesis on it, obtaining the following bound

$$\begin{aligned}
& & |\delta_{i,T_i(c)}| &= |\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} - \mu_i| & (49)\\
&[\textit{By inductive hypothesis}] & &= O\left(\frac{L^{D-1}K(D-1)\ln T_i(c) + L^{D-1}K^{D-1}}{T_i(c)}\right) & (50)\\
&[\lim_{m\to\infty}\tfrac{T_i(c(m))}{m} = \pi_0(i), \forall i \in \mathcal{B}_A] & &= O\left(\frac{L^{D-1}K(D-1)\ln(m\cdot\pi_0(i)) + L^{D-1}K^{D-1}}{m\cdot\pi_0(i)}\right) & (51)\\
&[\textit{Properties of logarithms}] & &= O\left(\frac{L^{D-1}K(D-1)\ln m + L^{D-1}K(D-1)\ln\pi_0(i) + L^{D-1}K^{D-1}}{m\cdot\pi_0(i)}\right) & (52)\\
&[L^{D-1}K(D-1)\ln\pi_0(i) \leq 0 \rightarrow \textit{removed}] & &= O\left(\frac{L^{D-1}K(D-1)\ln m + L^{D-1}K^{D-1}}{m\cdot\pi_0(i)}\right) & (53)\\
& & &= O\left(\frac{L^{D-1}KD\ln m - L^{D-1}K\ln m + L^{D-1}K^{D-1}}{m\cdot\pi_0(i)}\right) & (54)
\end{aligned}$$

Summing up over all bootstrapped actions i according to $\sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}|$ we obtain

$$\sum_{i\in\mathcal{B}_A}\pi_0(i)\cdot|\delta_{i,T_i(c)}| = O\left(\sum_{i\in\mathcal{B}_A}\pi_0(i)\frac{L^{D-1}KD\ln m - L^{D-1}K\ln m + L^{D-1}K^{D-1}}{m\cdot\pi_0(i)}\right) \tag{55}$$

$$= O\left(\sum_{i\in\mathcal{B}_A}\frac{L^{D-1}KD\ln m - L^{D-1}K\ln m + L^{D-1}K^{D-1}}{m}\right) \tag{56}$$

$$[\textit{Sum over L bootstrapped actions,}] \quad [\textit{no dependence on } i \textit{ in the sum}] \tag{57}$$

$$= O\left(L\cdot\frac{L^{D-1}KD\ln m - L^{D-1}K\ln m + L^{D-1}K^{D-1}}{m}\right) \tag{58}$$

$$= O\left(\frac{L^{D}KD\ln m - L^{D}K\ln m + L^{D}K^{D-1}}{m}\right) \tag{59}$$

The second term we consider is $p_{\overline{\mathcal{B}}}\cdot|\delta_n^*|$ which is related to the bias of the optimal non-bootstrapped action. We obtain the bound

$$p_{\overline{\mathcal{B}}}\cdot|\delta_n^*| = p_{\overline{\mathcal{B}}}|\mathbb{E}\left\{\bar{X}_{i^*,n}\right\} - \mu^*| \tag{60}$$

$$[\textit{By inductive hypothesis}] \quad = p_{\overline{\mathcal{B}}}\cdot O\left(\frac{L^{D-1}K(D-1)\ln n + L^{D-1}K^{D-1}}{n}\right) \tag{61}$$

$$[\lim_{m\to\infty}\tfrac{n(m)}{m} = p_{\overline{\mathcal{B}}}] \quad = p_{\overline{\mathcal{B}}}\cdot O\left(\frac{L^{D-1}K(D-1)\ln(p_{\overline{\mathcal{B}}}\cdot m) + L^{D-1}K^{D-1}}{p_{\overline{\mathcal{B}}}\cdot m}\right) \tag{62}$$

$$= O\left(\frac{L^{D-1}K(D-1)\ln(p_{\overline{\mathcal{B}}}\cdot m) + L^{D-1}K^{D-1}}{m}\right) \tag{63}$$

$$[\textit{Properties of logarithms}] \quad = O\left(\frac{L^{D-1}K(D-1)\ln p_{\overline{\mathcal{B}}} + L^{D-1}K(D-1)\ln(m) + L^{D-1}K^{D-1}}{m}\right) \tag{64}$$

$$[L^{D-1}K(D-1)\ln p_{\overline{\mathcal{B}}} \le 0 \to \textit{removed}] \quad = O\left(\frac{L^{D-1}K(D-1)\ln m + L^{D-1}K^{D-1}}{m}\right) \tag{65}$$

$$= O\left(\frac{L^{D-1}KD\ln m - L^{D-1}K\ln m + L^{D-1}K^{D-1}}{m}\right) \tag{66}$$

The third term we consider is $O\left(\frac{K(\ln m + N_0)}{m}\right)$, namely, the second part of the bias related to non-bootstrapped actions

$$O\left(\frac{K(\ln m + N_0)}{m}\right) = O\left(\frac{K\ln m + KN_0}{m}\right) \tag{67}$$

$$[\textit{With } N_0 = O(L^DK^{D-1})] \quad = O\left(\frac{K\ln m + L^DK^D}{m}\right) \tag{68}$$

Finally, we put together the three terms of Equations 59, 66, and 68 in Equation 48, obtaining a bound for the bias of the estimated expected payoff of the root of the tree having depth D, which is

$$\begin{aligned}
\left|\mathbb{E}\{\bar{X}_m\} - \sum_{i\in\mathcal{B}_A} \pi_0(i)\mu_i - p_{\overline{\mathcal{B}}}\cdot\mu^\star\right| &\le O\left(\frac{L^D K D \ln m - L^D K \ln m + L^D K^{D-1}}{m}\right) && (69)\\
&+ O\left(\frac{L^{D-1} K D \ln m - L^{D-1} K \ln m + L^{D-1} K^{D-1}}{m}\right) && (70)\\
&+ O\left(\frac{K \ln m + L^D K^D}{m}\right) && (71)\\
\text{[Rearranging terms]} \quad &= O\left(\frac{L^{D-1} K D \ln m + L^D K D \ln m + K \ln m}{m}\right) && (72)\\
&+ O\left(\frac{L^D K^D + L^{D-1} K^{D-1} + L^D K^{D-1}}{m}\right) && (73)\\
&+ O\left(\frac{-L^{D-1} K \ln m - L^D K \ln m}{m}\right) && (74)\\
\text{[Removing (positive) terms with smaller } D \text{ order]} \quad &= O\left(\frac{L^D K D \ln m + L^D K^D - L^{D-1} K \ln m - L^D K \ln m}{m}\right) && (75)\\
[-L^{D-1}K\ln m \le 0,\ -L^D K \ln m \le 0 \to \text{removed}] \quad &= O\left(\frac{L^D K D \ln m + L^D K^D}{m}\right) && (76)
\end{aligned}$$

which proves the induction. □

The third theorem provides, finally, a concentration bound that shows that the estimated optimal payoff concentrates quickly around its mean.

Theorem A.8 (Concentration bound for the mean.). *Fix an arbitrary $\delta > 0$ and let $\Delta_m = 9\sqrt{2p_{\overline{\mathcal{B}}} m \ln(4/\delta)} + C_p\sqrt{m\ln(2L/\delta)}\sum_{i\in\mathcal{B}_A}\sqrt{\pi_0(i)}$. Let $n_0 \in \mathbb{N}$ be such that $\sqrt{n_0} \ge O(K(C_p^2 \ln n_0 + N_0(1/2)))$, if $m \ge \frac{n_0}{p_{\overline{\mathcal{B}}}}$ then under Assumption A.5 the following bounds hold:*

$$\mathbb{P}\left\{m\bar{X}_m \ge m\mathbb{E}\left\{\bar{X}_m\right\} + \Delta_m\right\} \le \delta, \tag{77}$$

$$\mathbb{P}\left\{m\bar{X}_m \le m\mathbb{E}\left\{\bar{X}_m\right\} - \Delta_m\right\} \le \delta. \tag{78}$$

Proof. Let us consider the first bound, in Equation 77. We first split the probability in two parts, namely,

$$\begin{aligned}
&\mathbb{P}\left\{m\bar{X}_m \ge m\mathbb{E}\left\{\bar{X}_m\right\} + \Delta_m\right\} = && (79)\\
\text{Since } \bar{X}_m = \tfrac{c}{m}\cdot\bar{X}_c + \tfrac{n}{m}\cdot\bar{X}_n \quad &= \mathbb{P}\left\{c\bar{X}_c + n\bar{X}_n \ge \mathbb{E}\left\{c\bar{X}_c + n\bar{X}_n\right\} + \Delta_m\right\} && (80)\\
\text{Linearity of expectation and } \Delta_m := \Delta_c + \Delta_n \quad &= \mathbb{P}\left\{c\bar{X}_c + n\bar{X}_n \ge c\cdot\mathbb{E}\left\{\bar{X}_c\right\} + n\cdot\mathbb{E}\left\{\bar{X}_n\right\} + \Delta_c + \Delta_n\right\} && (81)\\
\text{Since } \mathbb{P}\{A+B\ge C+D\} \le \mathbb{P}\{A\ge C\} + \mathbb{P}\{B\ge D\} \quad &\le \mathbb{P}\left\{c\bar{X}_c \ge c\cdot\mathbb{E}\left\{\bar{X}_c\right\} + \Delta_c\right\} + \mathbb{P}\left\{n\bar{X}_n \ge n\cdot\mathbb{E}\left\{\bar{X}_n\right\} + \Delta_n\right\} && (82)\\
&= P_{\mathcal{B}} + P_{\overline{\mathcal{B}}}. && (83)
\end{aligned}$$

The part related to *non-bootstrapped* actions $P_{\overline{\mathcal{B}}} := \mathbb{P}\left\{n\bar{X}_n \ge n\cdot\mathbb{E}\left\{\bar{X}_n\right\} + \Delta_n\right\}$ can be bounded using Theorem 5 of (Kocsis, Szepesvari, et al. 2006). For an arbitrary $\delta_{\overline{\mathcal{B}}} = \delta/2$ and $\Delta_n = 9\sqrt{2n\ln(2/\delta_{\overline{\mathcal{B}}})} = 9\sqrt{2n\ln(4/\delta)}$ there exist a n_0 such that $\sqrt{n_0} \ge O(K(C_p^2 \ln n_0 + N_0(1/2)))$ and for any $n \ge n_0$, under Assumption A.5 the following bound holds:

$$P_{\overline{\mathcal{B}}} = \mathbb{P}\left\{n\bar{X}_n \geq n \cdot \mathbb{E}\left\{\bar{X}_n\right\} + \Delta_n\right\} \leq \delta_{\overline{\mathcal{B}}} = \delta/2 \tag{84}$$

We notice that Δ_n can also be expressed in terms of the total number of simulations m, since $\lim_{m\to\infty} \frac{n(m)}{m} = p_{\overline{\mathcal{B}}}$. In that case we have

$$\Delta_n = 9\sqrt{2p_{\overline{\mathcal{B}}} m \ln(4/\delta)}. \tag{85}$$

The probability related to *bootstrapped* actions $P_{\mathcal{B}} \coloneqq \mathbb{P}\left\{c\bar{X}_c \geq c \cdot \mathbb{E}\left\{\bar{X}_c\right\} + \Delta_c\right\}$ can be decomposed into probabilities for the single actions, as follows:

$$\begin{aligned}
P_{\mathcal{B}} &= \mathbb{P}\left\{c\bar{X}_c \geq c \cdot \mathbb{E}\left\{\bar{X}_c\right\} + \Delta_c\right\} && (86)\\
[\textit{Since } \bar{X}_c \coloneqq \tfrac{1}{c}\textstyle\sum_{i\in\mathcal{B}_A} T_i(c)\bar{X}_{i,T_i(c)}] \quad &= \mathbb{P}\left\{\sum_{i\in\mathcal{B}_A} T_i(c)\bar{X}_{i,T_i(c)} \geq \sum_{i\in\mathcal{B}_A} T_i(c)\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} + \Delta_c\right\} && (87)\\
[\Delta_c \coloneqq \textstyle\sum_{i\in\mathcal{B}_A}\Delta_{c_i}] \quad &= \mathbb{P}\left\{\sum_{i\in\mathcal{B}_A} T_i(c)\bar{X}_{i,T_i(c)} \geq \sum_{i\in\mathcal{B}_A} (T_i(c)\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} + \Delta_{c_i})\right\} && (88)\\
[\textit{Since } \mathbb{P}\{\textstyle\sum_i A_i \geq \sum_i B_i\} \leq \sum_i \mathbb{P}\{A_i \geq B_i\}] \quad &\leq \sum_{i\in\mathcal{B}_A} \mathbb{P}\left\{T_i(c)\bar{X}_{i,T_i(c)} \geq (T_i(c)\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} + \Delta_{c_i})\right\} && (89)
\end{aligned}$$

We consider each term $\mathbb{P}\left\{T_i(c)\bar{X}_{i,T_i(c)} \geq (T_i(c)\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} + \Delta_{c_i})\right\}$ in the sum separately. We observe that it corresponds to Equation 29 of Assumption A.5, substituting $T_i(c)$ to n and $\Delta_{c_i} = C_p\sqrt{T_i(c)\ln(\frac{1}{\delta_{\mathcal{B},i}})}$ to Δ_n with $\delta_{\mathcal{B},i} > 0, \forall i \in \mathcal{B}_A$. Also in this case Δ_{c_i} can be expressed in terms of m since $\lim_{m\to\infty} \frac{T_i(c(m))}{m} = \pi_0(i), \forall i \in \mathcal{B}_A$. In that case we have

$$\Delta_{c_i} = C_p\sqrt{\pi_0(i) m \ln(\frac{1}{\delta_{\mathcal{B},i}})}. \tag{90}$$

Therefore, $\mathbb{P}\left\{T_i(c)\bar{X}_{i,T_i(c)} \geq (T_i(c)\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} + \Delta_{c_i})\right\} \leq \delta_{\mathcal{B},i}$ by Assumption A.5 (proved above using the Hoeffding inequality). This guarantees that the estimated payoff of each bootstrapped action concentrates quickly around its mean, hence computing the sum over bootstrapped actions in Eq. 89 we obtain:

$$\begin{aligned}
P_{\mathcal{B}} &\leq \sum_{i\in\mathcal{B}_A} \mathbb{P}\left\{T_i(c)\bar{X}_{i,T_i(c)} \geq (T_i(c)\mathbb{E}\left\{\bar{X}_{i,T_i(c)}\right\} + \Delta_{c_i})\right\} && (91)\\
&\leq \sum_{i\in\mathcal{B}_A} \delta_{\mathcal{B},i} && (92)\\
[\delta_{\mathcal{B},i} \coloneqq \delta/(2L)] \quad &= \delta/2 && (93)
\end{aligned}$$

Notice that we selected $\delta_{\mathcal{B},i} \coloneqq \delta/(2L)$, so that $\delta_{\mathcal{B}} = \sum_{i\in\mathcal{B}_A} \delta_{\mathcal{B},i} = L\delta/(2L) = \delta/2$. To conclude the proof we substitute $\delta_{\mathcal{B},i}$ in Δ_{c_i} obtaining $\Delta_{c_i} = C_p\sqrt{\pi_0(i)m\ln(\frac{2L}{\delta})}$ and we sum up Δ_{c_i} over bootstrapped actions obtaining

$$\begin{aligned}
\Delta_c &= \sum_{i\in\mathcal{B}_A} \Delta_{c_i} && (94)\\
&= \sum_{i\in\mathcal{B}_A} C_p\sqrt{\pi_0(i) m \ln(\frac{2L}{\delta})} && (95)\\
&= C_p\sqrt{m\ln(2L/\delta)} \sum_{i\in\mathcal{B}_A} \sqrt{\pi_0(i)} && (96)
\end{aligned}$$

In conclusion

$$\mathbb{P}\left\{m\bar{X}_m \geq m\mathbb{E}\left\{\bar{X}_m\right\} + \Delta_m\right\} \leq P_{\mathcal{B}} + P_{\overline{\mathcal{B}}} \tag{97}$$
$$\leq \delta/2 + \delta/2 \tag{98}$$
$$= \delta \tag{99}$$

with

$$\Delta_m = \Delta_n + \Delta_c \tag{100}$$
$$= 9\sqrt{2p_{\overline{\mathcal{B}}} m \ln(4/\delta)} + C_p\sqrt{m \ln(2L/\delta)} \sum_{i \in \mathcal{B}_A} \sqrt{\pi_0(i)} \tag{101}$$

Equation 78 can be proved similarly.

□

B Details about Safe Policy Improvement

SPI was formalized on MDPs using different *percentile criteria* (Delage and Mannor 2010). A percentile criterion was first proposed in 2007 for exploration (Delage and Mannor 2007) and in 2008 for safety (Schneegass, Udluft, et al. 2008). SPIBB considers the *worst-case scenario* (Petrik et al. 2016), namely, $\pi_I = \arg\max_{\pi \in \Pi} \min_{M \in \Xi}(\rho(\pi, M) - \rho(\pi_0, M))$, where Ξ is the set of admissible MDPs $\Xi(M^{\mathcal{D}}, e) = \{M = \langle S, A, R, T, \gamma\rangle \mid \forall(s,a) \in S \times A, \|T(s,a,\cdot) - T^{\mathcal{D}}(s,a,\cdot)\|_1 \leq e(s,a)\}$. The MDPs in this set have a transition model T with L_1 distance from the transition model $T^{\mathcal{D}}$ estimated from data $\mathcal{D}$ smaller than the error $e(s,a)$. The error function $e : S \times A \longrightarrow \mathbb{R}$ is an arbitrary function representing the uncertainty over the estimated transition model $T^{\mathcal{D}}$. The optimization of π_I is, however, NP-hard.

C Supplementary Material About the Methodology

C.1 Hyperparameters of the Experiments

In Table 2, we describe the parameters of the experiments on scalability presented in Section 4.6. In Table 4, we describe the parameters used in the safety experiments presented in Section 4.7. In Table 5 we describe the parameters used in the ablation study presented in Section 4.8.

Table 2. Hyperparameters of the experiments on scalability described in Section 4.6.

Experiment	Scalability (see Figure 2)
Domain	single-agent SysAdmin with 4, 7, 10, 12, 13, 20, 35 machines
Number of datasets	20
$\|\mathcal{D}\|$	5000
$\mathbf{N}_\wedge$	5
Measures	$\bar{\rho}(\pi_I, M^*)$, time [s]
Number of runs	20 runs of 15 steps each
Number of MCTS simulations	4000
Baseline policy	sub-optimal ($p = 0.7$)
Experiment	Scalability (see Figure 3)
Domain	multi-agent SysAdmin with 4, 8, 10, 16 agents
Number of datasets	20
$\|\mathcal{D}\|$	10000, 50000, 100000, 200000, 500000
$\mathbf{N}_\wedge$	5
Measures	$\bar{\rho}(\pi_I, M^*)$, time [s]
Number of runs	20 runs of 25 steps each
Number of MCTS simulations	10000
Baseline policy	sub-optimal ($p = 0.7$)
Experiment	Scalability (see Figure 4)
Domain	multi-agent SysAdmin with 16 agents
Number of datasets	20
$\|\mathcal{D}\|$	10000, 50000, 100000, 200000, 500000, 1700000
$\mathbf{N}_\wedge$	5
Measures	$\bar{\rho}(\pi_I, M^*)$, time [s]
Number of runs	20 runs of 25 steps each
Number of MCTS simulations	10000
Baseline policy	random ($p = 0.5$) and sub-optimal ($p = 0.9$)

Table 3. Hyperparameters of the experiments on Safety described in Section 4.7.

Experiment	Safety (see Figure 5)
Domain	Wet-Chicken
Number of datasets	20
$\lvert\mathcal{D}\rvert$	100, 200, 500, 1000, 2000, 5000
$N_{\wedge}$	5
Measures	$\rho(\pi_I, M^*)$, 15% CVaR $\rho(\pi_I, M^*)$
Number of runs	policy evaluation
Number of MCTS simulations	10000
Baseline policy	sub-optimal ($p = 0.7$)
PPO	
Learning rate	[0.00001, 0.00003, 0.0001, **0.0003**]
Entropy coefficient	[0.0, 0.005, **0.01**]
Clip range	[**0.1**, 0.2, 0.3]
Batch size	[64, **128**, 256]
Number of epochs	[**4**, 10]
DQN	
Learning rate	[0.00001, 0.00003, **0.0001**, 0.0003]
Exploration fraction	[**0.1**, 0.3]
Batch size	[64, 128, **256**]
Buffer size	500000
Target update interval	[**100**, 500]
CQL	
Learning rate	[0.00001, 0.00003, 0.0001, **0.0003**]
Alpha	[**1**, 5, 10]
Batch size	[64, 128, **256**]
Q updated per step	[**200**, 500]
Target update interval	[**100**, 500]
Decision Transformer	
Context length	[1, 10, **20**, 80, 100]
Hidden size	[56, 64, 128, **256**, 512]
Number of transformer layers	[1, 2, 4, **8**]
Number of attention heads	[2, **4**, 8, 10]
Dropout rate	[0.0, 0.001, 0.01, **0.1**]
Learning rate	[0.00001, **0.0001**, 0.001, 0.01]
Batch size	[64, **128**, 256]
Number of epochs	[1, 3, 10, 20, **30**, 50, 80, 100]
MOReL	
Ensemble hidden sizes	[64, 128, 256, 512]
Ensemble learning rates	[**0.0001**, 0.001, 0.01]
Ensemble batch sizes	[64, **128**, 256]
Ensemble epochs list	[**5**, 10, 20, 50]
Ensemble sizes	[**2**, 4, 8]
PPO learning rate	[0.00001, 0.00003, 0.0001, **0.0003**]
PPO entropy coefficient	[0.0, 0.005, **0.01**]
PPO clip range	[**0.1**, 0.2, 0.3]
PPO batch size	[64, **128**, 256]
PPO number of epochs	[**4**, 10]

Table 4. Hyperparameters of the experiments on Safety described in Section 4.7.

Experiment	Safety (see Figure 7.a, b)
Domain	SysAdmin with 7 agents
Number of datasets	20
$\|\mathcal{D}\|$	5, 500, 5000
N$_\wedge$	50
Measures	$\rho(\pi_I, M^*)$, 15% CVaR $\rho(\pi_I, M^*)$
Number of runs	policy evaluation
Number of MCTS simulations	10000
Baseline policy	sub-optimal ($p = 0.7$)
Experiment	Safety (see Figure 8.a, b)
Domain	GridWorld 5×5
Number of datasets	20
$\|\mathcal{D}\|$	2, 10, 100, 1000, 10000
N$_\wedge$	20
Measures	$\rho(\pi_I, M^*)$, 15% CVaR $\rho(\pi_I, M^*)$
Number of runs	policy evaluation
Number of MCTS simulations	10000
Baseline policy	sub-optimal ($p = 0.7$)
Experiment	Safety (see Figure 9)
Domain	SysAdmin with 7 agents
Number of datasets	20
$\|\mathcal{D}\|$	5, 100, 1000, 5000, 10000, 100000
N$_\wedge$	50
Measures	$\rho(\pi_I, M^*)$, 15% CVaR $\rho(\pi_I, M^*)$
Number of runs	policy evaluation
Number of MCTS simulations	10000
Baseline policy	random ($p = 0.5$)

Table 5. Hyperparameters of the experiment described in Section 4.8.3.

Experiment	Ablation study (see Figure 10)
Domain	SysAdmin with 7 agents
Number of datasets	20
$\lvert\mathcal{D}\rvert$	5000
$\mathbf{N}_{\wedge}$	50
Measures	$\rho(\pi_I, M^*)$
Number of runs	20 of policy evaluation for each dataset size
Number of MCTS simulations	10000
Baseline policy	random ($p = 0.5$) and sub-optimal ($p = 0.9$)
Experiment	Ablation study (see Figure 11)
Domain	SysAdmin with 7 agents
Number of datasets	20
$\lvert\mathcal{D}\rvert$	5000
$\mathbf{N}_{\wedge}$	50
Measures	$\rho(\pi_I, M^*)$
Number of runs	20 of policy evaluation for each dataset and number of simulations
Number of MCTS simulations	100, 1000, 10000
Baseline policy	random ($p = 0.5$) and sub-optimal ($p = 0.9$)

D Experimental Setup

The experiments are conducted on a workstation equipped with a 13th Gen Intel Core i7-13700K CPU with 24 threads and 64.0 GB of RAM, running Ubuntu 22.04.5 LTS 64-bit as operating system. All code was implemented in Python, leveraging standard scientific computing libraries.

E Source Code

The full implementation of the proposed methods, along with the experimental setup used in this work, is available at GitHub

Received 23 June 2025; accepted 15 September 2025