2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Imitation-BT: Automating Behavior Tree Generation by Echoing Reinforcement Learning Agents

Shailendra Sekhar Bathula Ramviyas Parasuraman

***Abstract*— Understanding an autonomous agent's decision-making prowess is of paramount importance, as it increases trust and guarantees safety. Although agent policies learned through reinforcement learning (RL) and machine learning (ML) paradigms have demonstrated their dominance in various domains, they struggle with deployment in high-stakes environments due to their algorithmic opacity. A structured and transparent representation of a policy helps us understand, evaluate, and modify it if necessary. Due to their inherent reactivity, modularity, and transparent hierarchical representation, the Behavior Tree (BT) is an ideal solution to represent control policies. In this paper, we focus on building a knowledge representation transfer framework in which knowledge of trained RL agents is captured through imitation learning and then utilized to form a compact BT. Our primary focus is to retain maximum performance while improving the interpretability of the BTs. In combination with planning and learning, we automate the formation of a BT and offer an alternative, transparent architecture for policy representation. In an extensive analysis with a variety of gymnasium environments and the Robotics Package Delivery domain simulations, we demonstrate the significant performance retention capability and superior interpretability of the proposed Imitation-BT.**

***Index Terms*— Behavior Trees, Imitation Learning, Deep Reinforcement Learning, Explainable Reinforcement Learning**

I. Introduction

The selection of an optimal control architecture is pivotal in robotics, determining the safety, efficiency, and adaptability of autonomous systems [1]. Among the most versatile frameworks are Behavior Trees (BTs), which have been widely adopted from the video game industry [2], for their modularity and intuitive representation of complex tasks [3]. BTs uniquely unify diverse control paradigms, from Finite State Machines to Subsumption Architectures into a single, cohesive model, making them a powerful tool for designing sophisticated robot behaviors [4]. Furthermore, BTs have also recently been applied in the context of automated planning [5] and multi-robot systems [6].

The distinct advantages of the Behavior Tree (BT) architecture become apparent when contrasted with other common paradigms, as illustrated conceptually in Fig. 1. While a Deep Neural Network (Fig. 1, Left) offers powerful function approximation, its internal decision-making process is opaque. Conversely, a Decision Tree (Fig. 1, Middle) provides a transparent, rule-based policy, but its structure

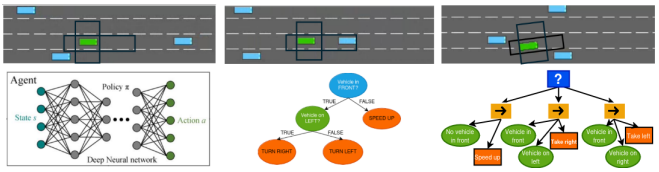

Fig. 1. A high-level comparison between different control architectures: **(Left)** Neural Networks are opaque outputting the primitive actions; **(Middle)** Decision Trees (DT) have granular rules to follow and directly map to the primitive actions based on the state but does not explain the reasons; **(Right)** Behavior Trees (BT) have a high-level reactive and priority-based architecture where the rationale of the decisions are interpretable.

is often monolithic and lacks a high-level behavioral hierarchy. The Behavior Tree (Fig. 1, Right) synthesizes those behaviours by organizing reactive rules within a modular and hierarchical framework that makes the policy/plan inherently interpretable. It is this unique combination of structure and transparency that motivates our choice of BTs as the target representation.

To imbue these traditionally manually crafted structures with intelligence, researchers increasingly integrate policies from learning communities like genetic programming [7], reinforcement learning (RL) [8], and multi-agent collaboration [9], allowing robots to learn optimal strategies through interactions. However, this integration, especially with RL, introduces a critical trade-off between performance and interpretability. RL policies, especially those using deep neural networks, operate as opaque black boxes [10]. This lack of transparency obscures the internal control flow, which hinders confident applications in safety-critical environments. Furthermore, RL policies suffer from a critical lack of adaptability; modifying their behavior requires data-hungry and computationally intensive retraining, making real-time policy adjustments infeasible [11]. Together, these issues of opacity and inflexibility undermine the modularity and trustworthiness that make BTs more attractive.

The field of explainable RL (XRL) has attempted to address the opacity problem by converting complex policies into more interpretable models [12]. While these methods offer a peek into the black boxes, an explainable yet monolithic policy remains difficult to modify, verify, or integrate into a broader system in which designers need to reuse or adapt specific skills.

To address these challenges, we introduce Imitation-BT, a novel knowledge transfer framework that creates a modular and interpretable Behavior Tree-based control architecture from an expert Deep Reinforcement Learning agent. Our framework offers an alternative for deployment, where we

School of Computing, University of Georgia, Athens, GA 30602, USA.
Corresponding author email: ramviyas@uga.edu
This research is supported by the Army Research Laboratory and was accomplished under DCIST Cooperative Agreement W911NF-17-2-0181.

uses imitation learning to distil expert(RL) strategies into an Action Template Library (ATL), on which it performs logic optimization to generate a transparent, structured BT that replicates the expert's performance. Furthermore, our framework is inherently modular, ensuring compatibility with any DRL algorithm, any decision tree induction method, or logic minimization technique. We evaluate our framework on classic gymnasium environments and a package delivery domain, demonstrating the maximal performance retention of our generated BTs and providing transparency.

II. Related Work and Background

A prominent line of work focuses on constructing Behavior Trees (BTs) from expert demonstrations, often using Decision Trees (DTs) as an interpretable intermediary [13]. The pioneering BT-Espresso [14] first established a technique for translating DT logic into a canonicalized behavior tree (CBT) [15]. This work was later extended by subsequent methods like RE:BT Espresso [16] and BT-Factor [17], which refined this process by introducing sophisticated logic reduction techniques to create more compact and expressive BTs. More recent work has been done to identify constraints by performing clustering on the demonstrations before giving it to a planner for BT construction [18]. A critical limitation of these behaviour cloning methods is the distributional shift problem [19]. States encountered during operation diverge from the expert demonstration distribution, leading to performance degradation, especially in dynamic environments.

There are hardly any methods that aim to automate BT construction from RL policies. The earliest method reported for discrete state-spaces is the QLBT approach [20], which clusters the states based on actions taken in the Q-table to form a BT. This method, however, is fundamentally limited to tabular RL algorithms, rendering it inapplicable to modern deep RL agents operating in complex state spaces. CQI-BT [21], on the other hand, uses a Conservative Q-Improvement Algorithm [22] to build BTs progressively through online interaction, by binning the continuous state space.

Another innovative approach, XRL-BT [23], constructs BTs from visual inputs by clustering pixels, and tracking a specific object of interest in these latent states. This method introduces a new form of opacity, as the learned latent states lack an inherent logical representation and limit generalization to dynamic environments.

The existing literature presents a clear trade-off: methods from expert demonstrations suffer from distributional shift, are not scalable to deep RL policies, or produce outputs that are not truly transparent.

Our framework, Imitation-BT, directly addresses this gap. We are the first to apply an interactive imitation learning pipeline [24] to translate the complex policies of pre-trained expert deep RL agents into a modular BT architecture. This approach guarantees high performance retention by learning from a capable expert while producing a final control policy that is transparent, robust, and easily modified by engineers to adapt to new constraints-a crucial requirement for deploying safe and reliable autonomous systems.

A. Behavior Trees

Formally, a Behavior Tree is a rooted, directed tree where each node represents a specific behavior or control flow mechanism [25]. Execution is driven by a tick signal sent from the root, which propagates down the tree. Each node performs its function and returns one of the three statuses to its parent: Success, Failure, or Running.

BTs consists of two main node types:

- Execution Nodes (Leaves): These interact with the environment.
  - **Action nodes** execute a task (e.g move forward).These return Success, Running or Failure.
  - **Condition nodes** check a state (e.g. is-obstacle-near?). These return Success or Failure.
- Control Flow Nodes (Internal): These direct the flow of ticks. The most common are:
  - **Sequence (–>)**, which executes children in order until one fails.
  - **Selector, also called Fallback (?)**, which executes children until one succeeds.
  - **Parallel ($\rightrightarrows$)** executes all of its children simultaneously. Its return status depends on a predefined policy, such as requiring a certain number of children to succeed. This is useful for tasks that involve concurrent actions.

In this work, we focus on distilling reactive policies, where actions are considered atomic and complete within a single timestep. For such actions, the node immediately returns either Success or Failure. The Running status is crucial for modeling durative actions, i.e., tasks that span multiple ticks (e.g., "move forward for 2 seconds"). As the scope of our framework is to translate reactive DRL experts, the Running status for Action nodes is not part of the distilled policies we generate.

B. Reinforcement Learning

Reinforcement learning is a framework for an agent to learn optimal behavior through trial-and-error interaction with an environment. The problem is formally described as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ), representing the state space, action space, transition dynamics, reward function, and a discount factor, respectively [26].

The agent's goal is to learn a policy $\pi(a|s)$, which is a mapping from states to actions, that maximizes the expected cumulative reward. In Deep Reinforcement Learning, this policy is represented and learned by neural networks, making it highly performant but difficult to interpret [10].

C. Imitation Learning

Imitation Learning (IL) is a paradigm where an agent learns to perform a task by mimicking demonstrations from an expert policy (π^*), rather than from an explicit reward signal [27]. This approach is valuable when designing a reward function that is difficult or when exploration is unsafe. The learner is given a dataset of the expert's state-action

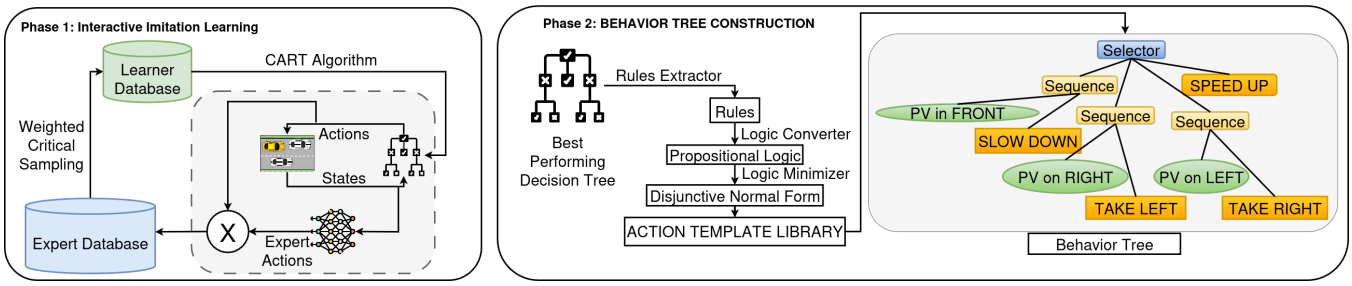

Fig. 2. Architecture of the proposed Imitation BT pipeline, categorized into multiple phases: 1.) **(Left)** Interactive Imitation Learning for Intermediary Decision Trees 2.) **(Right)** Interpretable Behavior Tree Construction with Logic Optimization from Decision Trees.

pairs and aims to find a policy (π) that replicates the expert's behavior.

The simplest form of IL is Behavior Cloning (BC), which treats the problem as a standard supervised learning task. It trains a policy, π_{BC}, to minimize a loss function, $\ell(s, \pi)$, on the static distribution of states visited by the expert, d^{π^*}:

$$\pi_{BC} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d^{\pi^*}} [\ell(s, \pi)] \tag{1}$$

However, this approach suffers from a critical issue known as distributional shift. Because the learner policy is imperfect, it will inevitably drift into states that were not present in the expert's original dataset ($d^{\pi^*} \neq d^{\pi_{BC}}$). In these novel states, the policy's behavior is undefined, leading to a cascade of compounding errors [28].

Formally, the performance gap between the learner and the expert can be bounded. If the learner policy π has an average error of ϵ on the expert's state distribution, its expected cumulative cost, $J(\pi)$, is bounded by [28]:

$$J(\pi) \leq J(\pi^*) + T^2 \epsilon \tag{2}$$

where T is the task horizon. This shows that the learner's performance degradation can grow quadratically with the length of the task, a direct consequence of the distributional shift. Mitigating this compounding error is the central challenge addressed by more advanced interactive IL algorithms.

III. Methodology

Our `Imitation-BT` framework translates a pre-trained, opaque expert RL policy into a transparent and modular Behavior Tree (BT). As illustrated in Figure 2, the pipeline consists of two primary phases: **(1) an interactive policy distillation phase** to learn an interpretable learner model from the expert, and **(2) a logic synthesis phase** to construct the final, compact BT from that model.

A. Problem Formulation

We assume access to a pre-trained expert RL policy π^*. Our objective is to generate a BT, denoted π_{BT}, that minimizes the performance gap relative to the expert's expected cumulative cost, $J(\pi_{BT}) \approx J(\pi^*)$, while being fully interpretable. The cost function is defined over a task horizon T and the state distribution d^π induced by a policy π:

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s \sim d_t^\pi} [L^\pi(s)] \tag{3}$$

where $L^\pi(s)$ is the expected immediate cost of policy π in state s.

B. Phase 1: Interactive Policy Distillation

The goal of this phase is to distil the expert policy into an interpretable Decision Tree (DT), π_θ, while mitigating the distributional shift problem inherent in simple Behavior Cloning. The objective is to find a policy $\hat{\pi}$ that minimizes the expected surrogate loss under its *own* induced state distribution:

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d^\pi} [\ell(s, \pi)] \tag{4}$$

1) The learner Policy and Weighted Resampling: To accelerate learning, we adopt the resampling heuristic from VIPER [24], which uses a modified loss function to prioritize critical states:

$$\ell_t(s, \pi) = V_t^{\pi^*}(s) - Q_t^{\pi^*}(s, \pi(s)) \tag{5}$$

As the DT's loss function is non-convex, we implement this heuristic by resampling data points (s, a) from our aggregated dataset D with a probability proportional to an importance weight $\tilde{\ell}(s)$, which is a convex upper bound of the VIPER loss:

$$p((s, a)) \propto \tilde{\ell}(s) \quad \text{where} \quad \tilde{\ell}(s) = V^{\pi^*}(s) - \min_{a \in \mathcal{A}} Q^{\pi^*}(s, a) \tag{6}$$

This forces the CART algorithm [29] to focus on correctly classifying high-stakes states. For policy-based experts, we approximate Q-values using $Q(s, a) \approx \log \pi^*(a|s)$.

2) The Learning Loop: The core of our distillation is an interactive learning loop inspired by DAgger [19]. At each iteration i, we execute a mixed policy to collect data:

$$\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i \tag{7}$$

where β_i is a mixing coefficient that determines the probability of using the expert's action. The collected state-action pairs are aggregated into a dataset D, on which the next learner DT, $\hat{\pi}_{i+1}$, is trained using the weighted resampling method above. This process repeats for N iterations.

C. Phase 2: Behavior Tree Construction via Logic Synthesis

Once distillation converges in the previous step, we extract the best performing DT, traverse through it to extract the decision paths, and represent them as a set of rules for

TABLE I
OVERVIEW OF THE COMPLEXITY AND VARIATION IN EXPERIMENTAL ENVIRONMENTS.

Environment	Domain	State Space	Action Space	Key Challenge
Cliff Walking	Toy Text	Discrete	Discrete	Safe navigation (large negative reward)
Frozen Lake	Toy Text	Discrete	Discrete	Safe navigation, sparse reward
Cart Pole	Classic Control	Continuous	Discrete	Balancing under termination conditions
Acrobot	Classic Control	Continuous	Discrete	Chaotic dynamics, sparse reward
Mountain Car	Classic Control	Continuous	Discrete	Sparse reward, requires momentum
Lunar Lander	Box2D	Continuous	Discrete	Stochastic, continuous dynamics
Highway	Autonomous Driving	Continuous	Discrete	Stochastic behavior of other agents
Package Delivery	Robotics	Discrete	Discrete	Domain-specific for BTs

each action, formatted as Disjunctive Normal Form (DNF) expressions in the Action Template Library (ATL).

Generation of BTs from the ATL can be formulated as follows. Let:

- $C = \{C_1, C_2, ..., C_m\}$ be the **conditions space**, where each C_i represents a binary condition.
- $A = \{A_1, A_2, ..., A_n\}$ be the **action space**, where each A_j represents a possible decision or outcome.
- $C_i^c = C \setminus C_i$ (Complement: condition C_i is false)
- P(C) = $(C \cap C^c)$ is the power set comprising all possible conditions.

Each state S is defined as a subset of conditions:

$$S \subseteq P(C) \tag{8}$$

meaning, that current conditions that are true for a state, cam be rewritten as

$$S_j = \bigcup_{i=1}^{k} C_i, \tag{9}$$

where every S_j is a union of arbitrary k number of conditions and k quantifies the complexity of the condition set S_j.

We define a decision function D as follows:

$$D(S_j) = A_j,. \tag{10}$$

This formulation lets us refactor conditions in each k, using techniques such as Espresso Minimizer [14], resulting in a more compact representation. Further minimization is possible by identifying conditions that consume other conditions, helping us approximate their bounds [16]. To further reduce the size of the resulting behavior tree (BT), we consider k as a heuristic:

$$h(S_j) = k_j \tag{11}$$

and sort all S_j such that:

$$S_1, S_2, ..., S_n \quad \text{s.t.} \quad h(S_1) \leq h(S_2) \leq \cdots \leq h(S_n) \tag{12}$$

Since S_j is now sorted by increasing heuristic complexity, the decision function is defined as:

$$D(S) = A_j \quad \text{if } S \subseteq S_j, \quad \text{where } S_1 \subseteq S_2 \subseteq \cdots \subseteq S_n \tag{13}$$

which ensures that simpler conditions (lower k_j) are evaluated first before more complex ones.

We define the action A_j corresponding to S_j with the highest k value as the default action, ensuring that a decision is always reached, which is observed in all RL agents.

$$A_{\text{default}} = A_n, \quad \text{where } k_n = \max_j k_j \tag{14}$$

Thus, if no other conditions satisfy $D(S)$, the system falls back to executing A_{default}.

In the resulting BT, the root node is a selector, with child nodes representing sequences for each non-default action. The default action A_{default} is directly connected to the root, bypassing explicit conditions S_{default}.

Since the decision function D ensures partitioning of the observation space and enforces strict rule-based execution, choosing the most frequent action as A_{default} minimizes BT size while maintaining correctness.

Each sequence node contains fallback conditions for cases where multiple S_j lead to the same A_j, structured as:

$$S_j = \bigcup_{i=1}^{k_j} C_i, \quad \text{where } k_1 \leq k_2 \leq \cdots \leq k_n. \tag{15}$$

The sequence terminates in an action node, ensuring execution adheres to the ordered decision logic while reducing redundancy in tree structure.

By prioritizing simpler decision paths and leveraging a default fallback mechanism, the resulting BT exhibits:

- **Fewer nodes** due to the early pruning of unnecessary evaluations in the process.
- **Compact structure** as default cases handles residual cases without additional branching.
- **Faster execution** since the most probable conditions are evaluated first during inference/run time of BTs.

In summary, our framework bridges the gap between planning-based BTs and learning-based policy representation through imitation learning. By translating RL policies into transparent BTs, we achieve control policies that are both safe and modular due to their interpretability and are interpretable to observe desired behaviors in safety-critical tasks.

IV. EXPERIMENTS AND EVALUATION

To evaluate the efficacy and robustness of our Imitation-BT framework, we conducted a series of experiments de-

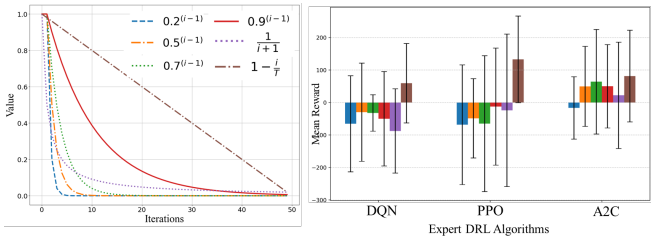

Fig. 3. Beta Analysis for the Lunar Lander domain: **(Left)** comparing the decay functions and **(Right)** their performance over multiple trials.

signed to answer two primary research questions:

1) Can our framework reliably replicate the performance of expert Deep Reinforcement Learning (DRL) agents across a diverse set of environments?
2) How does Imitation-BT compare to standard imitation learning baselines and prior work in Behavior Tree (BT) synthesis in terms of performance and the simplicity of the resulting policy?

A. Environments

We selected seven challenging environments from the OpenAI Gym toolkit [30] and one robotics-specific domain [31] to test our framework across a spectrum of challenges, including discrete and continuous state-action spaces, and deterministic and stochastic dynamics. The environments are summarized in Table I.

B. Baselines for Comparison

We compare the performance of Imitation-BT against several standard and state-of-the-art baselines:

- **Behavior Cloning (BC) [29]:** A standard supervised learning approach that learns from a static dataset of expert demonstrations.
- **DAgger [19]:** The seminal interactive imitation learning algorithm designed to mitigate distributional shift.
- **VIPER [24]:** An advanced imitation learning method that uses a Q-value-based heuristic, which we adapt in our framework.
- **BT-Espresso [14] & RE-BT-Espresso [16]:** State-of-the-art methods for generating BTs from offline expert demonstrations using logic optimization.

C. Implementation Details

- **Expert Training:** The expert DRL agents (DQN [10], PPO [32], A2C [33]) for each environment were trained using the Stable Baselines3 library [34]. Each agent used a policy network with two hidden layers of 64 units and was trained for 2 million time steps. All other hyperparameters were set to their default values.
- **Imitation Learning:** All imitation learning methods, including our Imitation-BT, were run for a total of 50 interactive iterations.
- **Software Stack:** Our implementation leverages Scikit-learn [35] for Decision Tree induction, PyEDA [36] for propositional logic operations, and Pytrees for the final BT construction.

D. Evaluation Metrics

We evaluate each method using two key metrics designed to measure both the efficacy and the interpretability of the learned policies:

- **Performance Retention (%):** To measure effectiveness, we calculate the final policy's average reward as a percentage of the original expert's average reward over multiple evaluation episodes. A score of 100% indicates that the distilled policy has successfully replicated the expert's performance.
- **Node Count:** To quantify interpretability, we measure the complexity of the final policy by its total number of nodes. For our method and other BT-based approaches, this is the total count of action, condition, and control flow nodes in the resulting Behavior Tree. A lower node count signifies a simpler and more easily interpretable policy.

E. Analysis on the β factor of Imitation Learning

The DAgger algorithm's performance can be highly sensitive to its mixing coefficient, β_i, which dictates the rate at which control is transferred from the expert to the learner. This is governed by the mixed policy at iteration i:

$$\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i \tag{16}$$

where π^* is the expert policy and $\hat{\pi}_i$ is the current learner policy. The standard approach uses an exponential decay schedule for the coefficient ($\beta_i = p^{i-1}$), which rapidly reduces expert supervision.

We hypothesized that for highly stochastic environments like Lunar Lander, this rapid hand-off might be suboptimal, as the learner is likely to make frequent errors during the critical initial learning phase. We posited that a more gradual linear decay schedule ($\beta_i = 1 - i/N$, where N is the total number of iterations) would yield a more robust final policy by providing extended expert guidance.

To validate this hypothesis, we conducted an empirical analysis on the Lunar Lander environment. We compared the performance of our proposed linear decay schedule against several exponential decay schedules, with the parameter p varied from 0.1 to 0.9. As shown in Figure 3, the results confirm our hypothesis. The linear decay schedule achieved the highest final performance, significantly outperforming all tested exponential schedules. This finding suggests that for complex, stochastic tasks, a slower and more consistent transfer of control is critical for achieving optimal performance in imitation learning.

F. Quantitative Analysis

Our experiments show that Imitation-BT consistently generates high-performing and compact Behavior Trees, outperforming existing methods, particularly in complex and stochastic environments.

1) Performance Retention: As shown in Figure 4, our Imitation-BT framework demonstrates superior performance retention across the majority of environments and expert types. In simpler, discrete domains like Cliff Walking, most

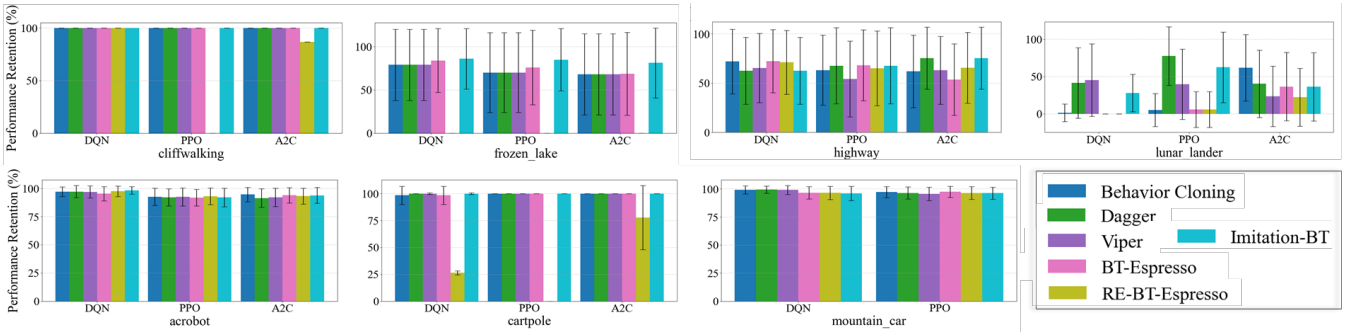

Fig. 4. Performance retention results of different compared approaches in seven different OpenAI Gym game domains (3x Classic Control, 2x Toy Text, 1x Box2D, 1x Highway) game domains. Different types of DRL policies (DQN, PPO, A2C) are used as the Expert inputs, where applicable.

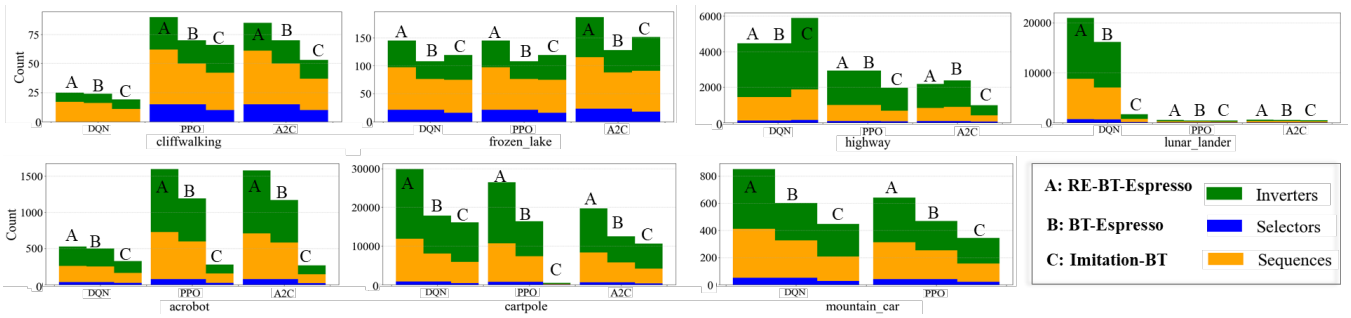

Fig. 5. Interpretability results of different compared approaches in seven different OpenAI Gym game domains (3x Classic Control, 2x Toy Text, 1x Box2D, 1x Highway) game domains.

methods, including Behavior Cloning, performed well. However, in more complex, continuous state spaces like Cart Pole and Acrobot, our interactive approach proved crucial for achieving near-perfect (up to 100%) performance replication where offline methods like RE:BT-Espresso struggled or failed entirely.

The most significant advantage of Imitation-BT was observed in the stochastic environments. While all imitation learning methods saw a performance drop in Lunar Lander and Highway due to the inherent difficulty of capturing stochastic policies from a limited dataset, our framework still consistently achieved the highest retention rates, especially with PPO and A2C experts in the Highway domain. This suggests that our weighted resampling and interactive learning are effective at capturing the most critical aspects of an expert's policy even under uncertainty.

2) Interpretability: In addition to performance, Imitation-BT produces significantly simpler and more interpretable policies. As illustrated in Figure 5, our method generated BTs with substantially fewer nodes than the state-of-the-art BT synthesis methods, with the most dramatic reductions—often by an order of magnitude—occurring in complex domains like Lunar Lander.

This compactness is a direct result of our logic optimization process. By prioritizing rules based on their simplicity and using a default action, our framework avoids creating redundant selector nodes. This structured approach ensures that the most common and simple conditions are evaluated first, leading to a BT that is not only smaller but also more computationally efficient at runtime.

G. Qualitative BT Analysis

Figure 6 representation of our Imitation-BT output compared to the other BT generators, such as the BT-Espresso [14] and RE:BT-Espresso [16], based on the same input DT. We can see that RE:BT Espresso with its float point reductions and increased expressivity offers a compact solution compared to BT-Espresso. Our Imitation BT design template eliminates conditions for our default action and further reduces the size of the obtain BT.

For the Frozen Lake environment, our Imitation-BT framework successfully generated a compact and optimal Behavior Tree that identifies both known paths to the goal state. As shown in Figure 7, the resulting BT effectively maps the discrete state space (grids 0–15) to the optimal actions. At the starting state (grid 0), the BT correctly identifies that both "right" represented by 2 and "down" represented by 1, are viable initial actions. The tree's logic further specifies the optimal action for subsequent states, such as choosing "right" for grids 1, 8, and 9 or anything above 12. The final BT structure consolidates all necessary rules and designates "down" as the default action, ensuring robust navigation. This learned policy accurately captures the two optimal paths to the goal: [0→4→8→9→10→14→15] and [0→1→2→6→10→14→15], demonstrating our framework's ability to distill a complete and effective strategy from the expert agent.

In the Mountain Car environment, which features a continuous state space (position, velocity) and three discrete actions, our Imitation-BT framework successfully distilled the expert's complex policy into a remarkably simple and intuitive rule. As shown in Figure 8, the generated Behavior Tree implements a clear, threshold-based strategy: if the car's

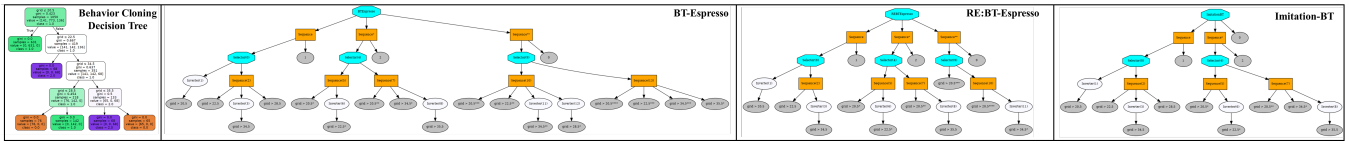

Fig. 6. The qualitative representation of the behavior trees generated in CliffWalking by BT-Espresso [14], RE:BT-Espresso [16], and our Imitation-BT. The leftmost figure on the left shows the input Decision Tree in the logic optimization process for comparison.

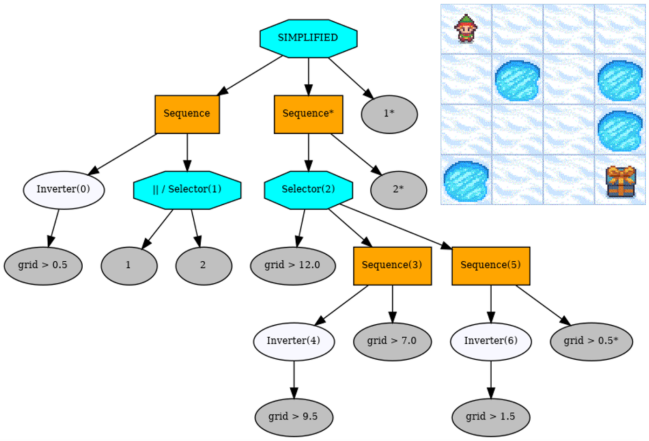

Fig. 7. Frozen Lake environment and its Imitation BT solution.

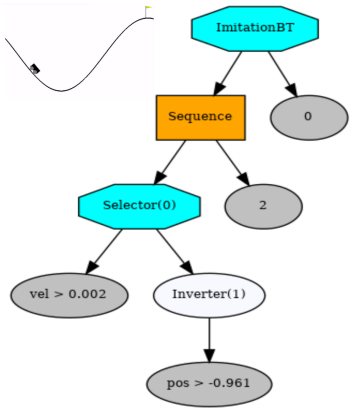

Fig. 8. Mountain Car environment and its Imitation BT solution.

position is greater than -0.96 or its velocity is greater than 0.002, the agent chooses "right" (action 2); otherwise, it chooses "left" (action 0).

This simple rule effectively captures the core momentum-building strategy required to solve the task. The agent learns to first move left (away from the goal) to build potential energy, and then apply thrust to the right once it has sufficient momentum to climb the hill. The discovery of such a clean and effective rule highlights our framework's ability to extract the essential logic from a "black-box" policy.

H. Case Study: Robot Package Delivery

To specifically evaluate the structural efficiency and logical consistency of our Imitation-BT framework, we conducted a comparative analysis against RE:BT-Espresso [16] in a robotics package delivery domain.

Environment: The simulation, illustrated in Figure 9, features an autonomous agent tasked with managing and delivering materials. The environment includes two loading

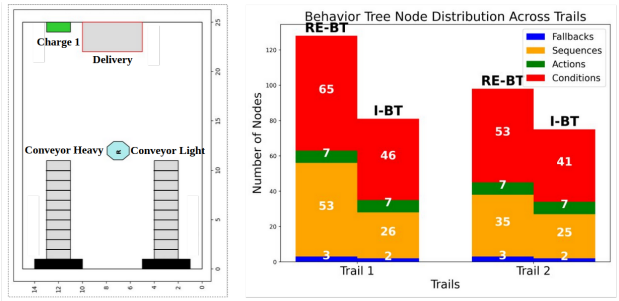

Fig. 9. Interpretability in the Robot Package Delivery Domain across two experiments **(Left)** Package Delivery Domain **(Right)** BT node distribution.

zones that randomly produce "heavy" or "light" materials, a depot, and a charging station. The agent must manage its battery, which depletes with time and actions (loading, unloading, moving), and make decisions based on the current world state. Both our method and the baseline were tasked with generating a BT to control the agent effectively.

Results: As shown in Figure 9, while both generated BTs achieved comparable task performance, our Imitation-BT framework produced a significantly more compact and interpretable policy. The BT generated by our method had a substantially lower node count compared to the one produced by RE:BT-Espresso. This reduction in complexity is a direct result of our heuristic-based design. By designating the action with the most complex conditions as the default action, we completely eliminate the need to represent its conditions in the tree. Since the rules extracted from the Decision Tree form a closed-state partition, we can confidently place this default action at the end of the root selector, guaranteeing completeness. This design choice highlights the effectiveness of our logic optimization, creating a simpler and more efficient BT structure without sacrificing performance.

V. Limitations and Future Works

Our framework has several important boundaries that open avenues for future research. First, the performance of the generated Behavior Tree is inherently bounded by the proficiency of the expert; our method is designed for high-fidelity replication , not outperformance. Future works could explore hybrid approaches that refine distilled BT with online learning. Second, the Decision tree induction process assumes a state space composed of well-defined, salient features and may struggle with raw, high-dimensional inputs like images. Integrating representation learning techniques to automatically discover these features is a promising direction. Furthermore, while DTs can handle continuous states, their axis-aligned splits may inefficiently approximate complex, non-linear decision boundaries. Finally, our current work does not address continuous action spaces, which could

lead to a combinatorial explosion of rules. Exploring alternative interpretable intermediary models remains an open and interesting question.

VI. Conclusion

In conclusion, we proposed a new imitation learning-based Behavior Tree (BT) construction framework, which enhances the transparency of opaque pre-trained Reinforcement Learning (RL) policies by converting RL policies into interpretable BTs. It integrates Interactive Imitation Learning to address covariate shifts, overcoming traditional behavior cloning issues. By translating decision trees into structured rules and then into canonicalized BT, we maintain the RL policy's stability and improve interpretability. Our heuristic-based default action selection further refines the BT, reducing complexity and clarifying the model. This approach effectively balances performance with transparency, providing insights into RL agents' decision-making processes.

References

[1] A. Ahmad and M. A. Babar, "Software architectures for robotic systems: A systematic mapping study," *Journal of Systems and Software*, vol. 122, pp. 16–39, 2016.

[2] M. Mateas and A. Stern, "A behavior language for story-based believable agents," *IEEE Intelligent Systems*, vol. 17, no. 4, pp. 39–47, 2002.

[3] P. Ogren, "Increasing modularity of uav control systems using computer game behavior trees," in *Aiaa guidance, navigation, and control conference*, p. 4458, 2012.

[4] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 372–389, 2017.

[5] S. Gugliermo, E. Schaffernicht, C. Koniaris, and F. Pecora, "Learning behavior trees from planning experts using decision tree and logic factorization," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3534–3541, 2023.

[6] S. S. OV, R. Parasuraman, and R. Pidaparti, "Impact of heterogeneity in multi-robot systems on collective behaviors studied using a search and rescue problem," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 290–297, IEEE, 2020.

[7] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of behavior trees for autonomous agents," *IEEE Transactions on Games*, vol. 11, no. 2, pp. 183–189, 2019.

[8] M. Kartasev, J. Salér, and P. Ögren, "Improving the performance of backward chained behavior trees that use reinforcement learning," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1572–1579, IEEE, 2023.

[9] S. S. O. Venkata, R. Parasuraman, and R. Pidaparti, "Kt-bt: A framework for knowledge transfer through behavior trees in multirobot systems," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 4114–4130, 2023.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[11] J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson, "A survey of meta-reinforcement learning," *arXiv preprint arXiv:2301.08028*, 2023.

[12] E. Puiutta and E. M. Veith, "Explainable reinforcement learning: A survey," in *International cross-domain conference for machine learning and knowledge extraction*, pp. 77–95, Springer, 2020.

[13] I. Sagredo-Olivenza, P. P. Gómez-Martín, M. A. Gómez-Martín, and P. A. González-Calero, "Trained behavior trees: Programming by demonstration to support ai game designers," *IEEE Transactions on Games*, vol. 11, no. 1, pp. 5–14, 2019.

[14] K. French, S. Wu, T. Pan, Z. Zhou, and O. C. Jenkins, "Learning behavior trees from demonstration," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 7791–7797, IEEE, 2019.

[15] B. Banerjee, "Autonomous acquisition of behavior trees for robot control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3460–3467, 2018.

[16] A. Wathieu, T. R. Groechel, H. J. Lee, C. Kuo, and M. J. Matarić, "Re:bt-espresso: Improving interpretability and expressivity of behavior trees learned from robot demonstrations," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 11518–11524, 2022.

[17] S. Gugliermo, E. Schaffernicht, C. Koniaris, and F. Pecora, "Learning behavior trees from planning experts using decision tree and logic factorization," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3534–3541, 2023.

[18] O. Gustavsson, M. Iovino, J. Styrud, and C. Smith, "Combining context awareness and planning to learn behavior trees from demonstration," in *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pp. 1153–1160, 2022.

[19] S. Ross, G. Gordon, and J. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," *Journal of Machine Learning Research - Proceedings Track*, vol. 15, 11 2010.

[20] R. Dey and C. Child, "Ql-bt: Enhancing behaviour tree design and implementation with q-learning," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, IEEE, 2013.

[21] C. Zhao, C. Deng, Z. Liu, J. Zhang, Y. Wu, Y. Wang, and X. Yi, "Interpretable reinforcement learning of behavior trees," in *Proceedings of the 2023 15th International Conference on Machine Learning and Computing*, ICMLC '23, (New York, NY, USA), p. 492–499, Association for Computing Machinery, 2023.

[22] A. M. Roth, N. Topin, P. Jamshidi, and M. Veloso, "Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy," *arXiv preprint arXiv:1907.01180*, 2019.

[23] K. Wan, Y. Liu, H. Liu, and X. Xu, "Unraveling explainable reinforcement learning using behavior tree structures," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6465–6469, IEEE, 2024.

[24] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 2499–2509, Curran Associates Inc., 2018.

[25] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[26] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *Robotica*, vol. 17, no. 2, pp. 229–235, 1999.

[27] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[28] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 661–668, PMLR, 13–15 May 2010.

[29] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[31] https://github.com/jstyrud/WASP-CBSS-BT.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[33] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[34] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[36] C. Drake, "Pyeda: Data structures and algorithms for electronic design automation.," in *SciPy*, pp. 25–30, 2015.