Lightweight and Faithful Visual Condition Checking in Behavior Trees via Expert-Regularized Reinforcement Learning

Hyosik Moon and Eldan Cohen
Department of Mechanical and Industrial Engineering
University of Toronto, Toronto, Canada
hyosik.moon@mail.utoronto.ca, eldan.cohen@utoronto.ca

Abstract

Behavior trees provide a transparent and modular structure for encoding expert-designed policies, enabling interpretable decision-making in complex tasks. Yet, applying behavior trees to high-dimensional perceptual inputs such as images or language is challenging as defining symbolic predicates over raw perceptual data is non-trivial. While state-of-the-art large multimodal models (such as vision-language models) can overcome this issue by utilizing natural language queries over perceptual inputs, they incur high computational cost, making them unsuitable for many applications. Imitation learning offers a way to distill these expert models into compact models, though it requires extensive supervision. In contrast, reinforcement learning reduces the need for costly supervision but risks misalignment of condition nodes with their intended semantics as well as poor credit assignment. To address these challenges, we introduce **CERL** (Condition-node Expert-regularized Reinforcement Learning), a framework that leverages expert-regularized reinforcement learning to preserve semantic faithfulness, while employing a factorized policy that aggregates sequential condition-node decisions into a single decision unit to alleviate credit assignment challenges. Experiments across seven tasks from the GymCards, FrozenLake, and BabyAIText suites demonstrate that our framework outperforms pure imitation learning or reinforcement learning baselines, retains strong agreement with expert decisions, and achieves substantial gains in inference speed and model size over expert models. Our implementation is available in https://github.com/HyosikMoon/CERL.

1 Introduction

Behavior Trees (BTs) (Colledanchise and Ögren, 2018; Iovino et al., 2022) are widely valued for their transparency, modularity, and reusability, and they are commonly used to encode expert-designed policies in robotics (Ögren and Sprague, 2022), game AI (Marcotte and Hamilton, 2017), and autonomous systems (Hu et al., 2021). Despite these strengths, their applicability to environments with rich perceptual inputs, such as images or natural language, remains limited. Standard BTs are inherently designed to operate on explicit, semantically meaningful features, yet extracting such features from high-dimensional sensory data at each step of a decision sequence is non-trivial. As a result, most BT applications rely on hand-crafted symbolic conditions or carefully engineered features, and only a few recent studies have attempted to extend BTs to perceptual inputs, either through feature engineering (Colledanchise et al., 2018) or by leveraging large vision–language models (VLMs) for condition evaluation (Wake et al., 2025).

In particular, VLMs (Ghosh et al., 2024) enable the evaluation of natural language condition queries directly on raw perceptual inputs, avoiding manual feature engineering. However, relying on a VLM for every decision during sequential decision-making is often impractical due to computationally expensive inference and the high cost of expert queries (e.g., API calls). A potential workaround is to utilize imitation learning (IL) (Zare et al., 2024) to train a compact condition-node policies based on VLM-labeled condition evaluation data, which can significantly reduce inference costs. Nonetheless, applying IL in the context of BTs poses significant challenge: IL methods typically require large amounts of labeled data, and in BTs this demand is further amplified because each environment action depends on multiple condition-node decisions along the traversal. As a result, IL can still be expensive for applications with limited budgets.

An alternative to IL is to use reinforcement learning (RL) to train compact condition-node policies directly from task completion rewards, thereby avoiding expensive VLM-based labeling. However, this approach faces two key challenges. First,

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 41764–41799

optimizing solely based on rewards can cause semantic drift, where condition-node outputs diverge from their intended semantics and instead prioritize reward-maximizing behaviors (Skalse et al., 2022). This undermines the original goal of BTs, transparent and interpretable decision logic through alignment with expert-designed conditions, since the learned nodes may no longer faithfully represent their intended meaning. Second, in sequential structures such as BTs, RL faces credit assignment challenges in sparse-reward settings, because each environment action is determined by a sequence of condition-node decisions, making it difficult to assign credit to these individual decision states.

Our framework addresses the first challenge – semantic drift – by incorporating expert regularization during RL, using a limited set of labeled expert decisions to preserve alignment with the intended conditions and prevent the learned policy from prioritizing reward at the cost of breaking BT semantics. The second challenge – credit assignment in BTs – is alleviated via a theoretically-justified factorized policy that aggregates condition-node decisions for an action as a single decision unit, improving learning signal propagation. Building on these principles, we introduce **CERL** (Condition-node Expert-regularized Reinforcement Learning), a framework that trains condition nodes directly from perceptual inputs through expert-regularized reinforcement learning, ensuring semantic faithfulness and robust learning under sparse rewards.

Our contributions are as follows: (1) we present a novel Markov Decision Process formulation and policy gradient derivation for RL-based training of condition nodes in BT; (2) we propose a unified framework that integrates IL and RL via expert regularization. Our method uses a limited set of expert labels (e.g., from a large VLM) both to initialize and continuously regularize the RL-based training of condition nodes, reducing expert labeling costs while preserving faithfulness to expert decisions; (3) we show that our method substantially improves over IL or RL alone, while maintaining high degree of faithfulness to the expert's decisions, and achieving fast inference with models orders of magnitude smaller and faster than the expert across GymCards, FrozenLake, and BabyAIText suites.

2 Preliminaries

Behavior Tree (BT) is a hierarchical control structure with a root node and a set of internal control flow nodes and execution (leaf) nodes (Colledanchise and Ögren, 2018). Control proceeds through a recursive "tick" process starting from the root, with each node returning one of three statuses: SUCCESS, FAILURE, or RUNNING. Control flow nodes determine traversal: a *Sequence* succeeds only if all children succeed, a *Fallback* (or *Selector*) succeeds if any child succeeds, and a *Parallel* ticks all children simultaneously and returns SUCCESS or FAILURE once a predefined threshold of children has succeeded or failed. Execution nodes interact directly with the environment: *Condition* nodes evaluate predicates and return SUCCESS or FAILURE, while *Action* nodes perform operations that may return RUNNING while ongoing and eventually return SUCCESS or FAILURE. Unlike control flow nodes, which are represented by symbolic operators (e.g., $\rightarrow$: *Sequence*, ? : *Selector*, $\rightrightarrows$: *Parallel*), execution nodes are conventionally denoted by shapes, ellipses for conditions and rectangles for actions. This modular, compositional structure makes BTs human readable and reusable, but standard BTs rely on symbolic or hand-engineered condition nodes, limiting their use in high-dimensional perceptual domains such as vision or language.

Imitation learning (IL) seeks to train a policy π that replicates the behavior of an expert π^*, typically from a dataset of demonstrations $\mathcal{D} = \{(s, a)\}$. The simplest form, behavioral cloning (Billard et al., 2008; Argall et al., 2009), treats IL as supervised learning by minimizing prediction loss between π and π^*. However, such offline training suffers from distributional shift: when deployed, the learned policy may visit states not covered in $\mathcal{D}$, compounding errors over time. Iterative IL methods, such as the DAgger (Ross et al., 2011) family of algorithms (Kelly et al., 2019; Hoque et al., 2022), address this issue by collecting expert labels on learner-visited states to mitigate compounding errors. While these methods reduce compounding errors, they require extensive expert queries during training, which can be prohibitively costly in practice.

Reinforcement learning (RL) provides a framework for learning policies via interaction with an environment, typically modeled as a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $P(s'|s, a)$ the transition dynamics, $R(s, a)$ the reward function, and $\gamma \in (0, 1)$ the discount factor. A stochastic policy $\pi(a|s)$ defines a distribution over actions given a state, with the objective of maxi-

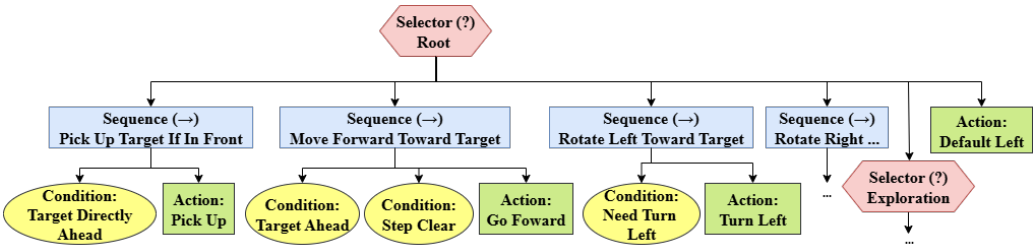


Figure 1: Example behavior tree (BT) for the *BabyAIText Pickup* task. Elliptical leaf nodes represent natural-language condition checks, and rectangular leaf nodes denote primitive actions. Internal control-flow nodes (e.g., Sequence and Selector) determine traversal. The complete tree can be found in Appendix A.8.

mizing the expected discounted return, $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$, where $\tau = (s_0, a_0, \ldots, s_T)$ denotes a trajectory. The policy gradient theorem (Sutton et al., 1999) gives

$$\nabla_\theta J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\left(\sum_{t=0}^{T}\gamma^t r_t\right)\nabla_\theta \log \pi(\tau)\right].$$

In practice, the return is replaced by variance-reduced estimators such as advantage functions; for example, Generalized Advantage Estimation (GAE) (Schulman et al., 2015) is commonly used in policy optimization methods such as Proximal Policy Optimization (PPO) (Schulman et al., 2017).

3 Problem Setup

We consider a behavior tree that encodes expert-designed policy for a given task, where condition nodes are expressed in natural language and operate on visual perceptual inputs. Our goal is to train compact models to evaluate these condition nodes so that they remain faithful to an expert's decisions. To achieve that, we assume access to such an expert (or a high-quality expert model, e.g., a large VLM) that can reliably evaluate condition nodes, i.e., judge whether the natural language query associated with each condition node is satisfied by the perceptual input. However, supervision queries are expensive and we have a limited budget for expert supervision. Furthermore, we focus on sparse-reward environments, where feedback is provided only at the end of each episode.

Example 1 (Behavior Tree Execution with Textual Conditions)**.** Figure 1 illustrates a simplified BT for the *BabyAIText Pickup* environment, where condition nodes correspond to natural language queries like *"Is the target object directly in front of the agent?"*. At each step during execution, the BT traverses from the root to a leaf and selects a primitive action. The accuracy of condition-node evaluations on the perceptual inputs determines whether the BT successfully executes the intended policy.

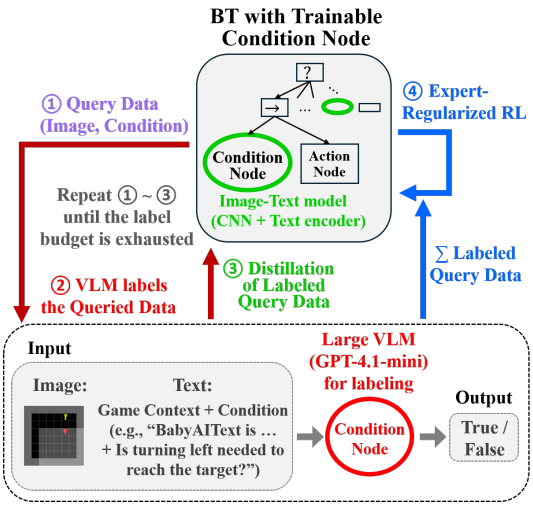


Figure 2: Overview of our framework. The compact model is trained with reinforcement learning under expert regularization, after being initialized with expert labels from a VLM.

Finally, to ensure compatibility with standard MDP formulation, we assume that at each environment step, the BT executes a complete root-to-leaf traversal and produces exactly one primitive action, with no dead ends.

4 Methodology

We propose a framework that enables lightweight and faithful visual condition checking in BTs via expert-regularized RL (Figure 2). Our approach unifies IL and RL by simultaneously optimizing condition nodes to align with limited expert supervision and to maximize task reward. Expert labels are used for parameter initialization as well as for regularization throughout training, to preserve semantic alignment with expert decisions. At each environment step, the sequence of condition-node

decisions is aggregated into a single decision unit corresponding to the chosen action, alleviating the credit assignment challenges described above. This unified process reduces the need for costly expert queries while maintaining faithfulness to expert decisions.

4.1 MDP Formulation and Policy Gradient

We formulate the decision-making process under a BT as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space consisting of high-dimensional perceptual observations (images), $\mathcal{A}$ is the set of primitive actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is the discount factor. We adopt the infinite-horizon discounted formulation ($\gamma \in (0, 1)$) for theoretical clarity, while our experiments are conducted in finite-horizon episodic tasks.

At each step t, the BT $\mathcal{T}$ is traversed from the root to a leaf, producing a sequence of condition-node decisions $\mathbf{y}_t = (y_{t,1}, \ldots, y_{t,n_t})$, where each $y_{t,i} \in \{0, 1\}$ is the binary outcome of the i-th condition node at time t, and n_t is the number of condition nodes evaluated along the traversal. The condition nodes are modeled using a shared binary classifier that receives the current perceptual state s_t (an image) together with the condition node's natural language query q_i. This classifier, implemented as a lightweight neural network, outputs the probability of the decision $y_{t,i}$ given the image–query pair (s_t, q_i), denoted $q_\theta(y_{t,i} \mid s_t, q_i)$, serving as a compact surrogate for the VLM expert. We note that the underlying MDP state space consists solely of images and the queries that represent the condition nodes are part of the fixed BT structure rather than the environment. Importantly, these queries are hand-designed, reflecting the desired behavior for a given task. Given a full decision sequence $\mathbf{y}_t$, it can then be mapped to a primitive action based on the BT.

To formalize this traversal, we use $p(\mathbf{y}_t \mid s_t)$ to denote the probability of selecting the decision sequence $\mathbf{y}_t \in \{0, 1\}^{n_t}$ at a given state $s_t \in \mathcal{S}$. Each sampled sequence $\mathbf{y}_t$ deterministically maps to a primitive action via the BT structure, $\mathcal{T} : \mathcal{Y} \to \mathcal{A}$. The probability distribution p naturally *induces* an action policy, where the probability of selecting an action a_t is given by the sum of probabilities of all decision sequences that map to it:

$$\pi(a_t \mid s_t) = \sum_{\mathbf{y}_t : \mathcal{T}(\mathbf{y}_t) = a_t} p(\mathbf{y}_t \mid s_t). \quad (1)$$

Proposition 1 then shows that this induced policy π is a valid action distribution, i.e., it is normalized and non-negative for all actions $a_t \in \mathcal{A}$.[1]

Proposition 1 (Distribution over Decision Sequences Induces a Valid Action Policy)**.** *Let $p(\mathbf{y}_t \mid s_t)$ be a normalized probability distribution over decision sequences (i.e., $\sum_{\mathbf{y}_t} p(\mathbf{y}_t \mid s_t) = 1$). Assume that the BT $\mathcal{T}$ is deterministic, and for every sequence $\mathbf{y}_t$ with $p(\mathbf{y}_t \mid s_t) > 0$, the BT returns exactly one primitive action $a_t = \mathcal{T}(\mathbf{y}_t) \in \mathcal{A}$. Define the induced action policy*

$$\pi(a_t \mid s_t) = \sum_{\mathbf{y}_t : \mathcal{T}(\mathbf{y}_t) = a_t} p(\mathbf{y}_t \mid s_t). \quad (2)$$

Then $\pi(a_t \mid s_t)$ is a valid probability distribution over $\mathcal{A}$.

While Proposition 1 establishes that the distribution p over decision sequences $\mathbf{y}_t$ induces a valid probability distribution over actions, directly computing $\pi(a_t \mid s_t)$ is inefficient since it requires summing over all decision sequences that map to each action. This issue is especially relevant for policy-gradient optimization, where we need to estimate expectations of functions (e.g., reward or advantage functions) by sampling from the action distribution. Proposition 2 shows that these action-level expectations can instead be written as expectations over the sequence distribution $p(\mathbf{y}_t \mid s_t)$, which will allow us to estimate them by sampling a single root-to-leaf path and using $a_t = \mathcal{T}(\mathbf{y}_t)$, without explicitly forming $\pi(a_t \mid s_t)$.

Proposition 2 (Expectation preservation under deterministic mapping)**.** *Let $a_t = \mathcal{T}(\mathbf{y}_t)$ for a deterministic mapping $\mathcal{T}$, and let $p(\mathbf{y}_t \mid s_t)$ be a valid distribution over decision sequences. Then, for any function f over actions (e.g., reward or advantage functions),*

$$\mathbb{E}_{a_t \sim \pi(\cdot|s_t)}[f(a_t)] = \mathbb{E}_{\mathbf{y}_t \sim p(\cdot|s_t)}[f(\mathcal{T}(\mathbf{y}_t))]. \quad (3)$$

Although Proposition 2 guarantees that expectations over actions can be replaced by expectations over decision sequences $\mathbf{y}_t$, directly modeling the full distribution $p(\mathbf{y}_t \mid s_t)$ is not practical. An autoregressive formulation could in principle capture conditional dependencies across condition nodes, but it would require sequential modeling along the path. This would undermine the interpretability of the learned BTs, since a widely recognized structural property of interpretable models is modularity

[1] All proofs appear in Appendix A.

(or composability) (Murdoch et al., 2019), which requires that the components of the model can be interpreted independently of each other. Therefore, we adopt a conditional independence assumption, factorizing condition-node decisions as independent given their image–query pairs:

$$p(\mathbf{y}_t \mid s_t) = \prod_{i=1}^{n_t} q_\theta(y_{t,i} \mid s_t, q_i), \tag{4}$$

where n_t is the number of condition nodes traversed at step t, $y_{t,i} \in \{0, 1\}$ denotes the binary decision at the i-th condition node, and (s_t, q_i) denotes the corresponding image–query pair. Recall that $q_\theta(y_{t,i} \mid s_t, q_i)$ denotes the probability that condition node i outputs $y_{t,i}$ given its image–query input, and it is implemented via a neural classifier parameterized by θ. Consequently, the sequence-level probability p and the action policy π are not parameterized directly, but are instead induced through q_θ and the deterministic BT mapping.

Under the factorization in Eq. (4), an action can be sampled by traversing the BT from root to leaf, sampling node-level decisions from q_θ and mapping the resulting sequence to a primitive action. This requires evaluating the nodes encountered along a single traversal, $O(n_{\max})$ per step (where $n_{\max}$ is the maximum depth of the BT), whereas explicitly computing the full action distribution would require aggregating over all possible root-to-leaf decision sequences, whose number can grow exponentially with the tree's branching structure.

Having established both the validity of the induced action distribution (Proposition 1) and the expectation preservation under deterministic mapping (Proposition 2), we now derive the policy gradient under the sequence distribution p, which factorizes into node-level distributions q_θ. Theorem 1 shows that the gradient naturally decomposes into condition-node log-probabilities, making optimization tractable.

Theorem 1 (Policy gradient for BT-driven policies)**.** *Let p denote the distribution over BT decision sequences, which induces the valid action policy π as established in Proposition 1. Using the expectation-preservation property of Proposition 2, the policy gradient, $\nabla_\theta J(\theta)$, is*

$$\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\sum_{i=1}^{n_t} A_t \nabla_\theta \log q_\theta(y_{t,i} \mid s_t, q_i)\right], \tag{5}$$

where A_t is any valid environment-time advantage, and the expectation is over trajectories τ sampled from the environment distribution induced by π.

In practice, we optimize a surrogate objective using PPO, maximizing $J_{\mathrm{PPO}}(\theta)$:

$$\mathbb{E}_t\Big[\min\Big(\rho_t(\theta)A_t, \mathrm{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)A_t\Big)\Big], \tag{6}$$

where the importance sampling ratio is defined as

$$\rho_t(\theta) := \frac{\prod_{i=1}^{n_t} q_\theta(y_{t,i} \mid s_t, q_i)}{\prod_{i=1}^{n_t} q_{\theta_{\mathrm{old}}}(y_{t,i} \mid s_t, q_i)}. \tag{7}$$

The advantage estimator A_t is computed using Generalized Advantage Estimation (Schulman et al., 2015). The full PPO loss combines the clipped surrogate in Eq. (6) with the importance sampling ratio defined over decision sequences (Eq. (7)), plus a value loss based on the state s_t and an entropy bonus computed over the node-level outputs $q_\theta(y_{t,i} \mid s_t, q_i)$, following the standard PPO.

Credit assignment under sparse rewards. Our factorized distribution treats all condition-node decisions at state s_t as a single decision sequence $\mathbf{y}_t = (y_{t,1}, \ldots, y_{t,n_t})$, which deterministically induces a primitive action via the BT. The update therefore uses a single environment-time advantage A_t applied to the log-probability, $\nabla_\theta \log p(\mathbf{y}_t \mid s_t)\, A_t$. We note that an alternative, simpler MDP formulation could treat each condition-node decision as a separate step, with states represented as pairs (s_t, q_i), expanding one environment step into $(t, 1), \ldots, (t, n_t)$. Rewards are then observed only after the final decision (t, n_t), which in sparse-reward settings is typically zero until episode termination. As a result, credit must propagate backward across $n_t - i$ intra-step links, causing the learning signal for early decisions (t, i) to be geometrically attenuated (approximately $\propto \gamma^{n_t-i}\lambda^{n_t-i}$, with discount γ and GAE parameter λ (Schulman et al., 2015)). By aggregating decisions within each environment step, our factorized formulation preserves a concentrated learning signal at the environment timescale and avoids inflating the critic's input domain from s_t to the expanded set of pairs (s_t, q_i) for all traversed condition nodes $i = 1, ..., n_t$ at each step, reducing the number of decision states evaluated. We empirically compare our formulation against this simpler baseline.

4.2 Expert-Regularized RL Training

RL with Expert Regularization. We optimize the policy via expert-regularized RL, using PPO

with Expert Regularization (ER) derived from cross-entropy against expert labels. At each iteration, trajectories are collected by executing the current policy through the BT in the environment. The PPO loss is computed from randomly sampled minibatches, using advantage estimates based on GAE (Schulman et al., 2015) and the importance sampling ratio (Eq. (7)); the value function is simultaneously updated to predict returns, as in standard PPO. To maintain alignment with expert knowledge, we also sample minibatches from the expert dataset $\mathcal{D}_{\text{sup}}$ at every parameter update step and compute the ER loss between the current policy's predictions and the corresponding precomputed VLM labels. The total loss combines the PPO and ER objectives, weighted by a regularization coefficient. The full training procedure is provided in Algorithm 1 (Appendix A.2).

Imitation Learning Initialization. We initialize the policy using IL, employing either Behavior Cloning (BC) or DAgger under a given labeling budget. For BC, we train on expert-labeled, mission-successful episodes using a cross-entropy loss, with early stopping based on an 8:1:1 train/-validation/test split; the best checkpoint is used to initialize RL. For DAgger, we adopt a fixed-budget, chunked aggregation scheme: at each iteration, we (i) collect a fixed quota of new labels by rolling out a mixture policy $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$, where π^* is the expert and $\hat{\pi}_i$ is the current learner, with β_i decayed according to an exponential schedule ($\beta_i = \beta_0 \cdot \alpha^i$), and then (ii) perform a single supervised update on the aggregated dataset, using the same cross-entropy objective as in BC. Unlike BC, which restricts training to successful expert-labeled episodes, DAgger aggregates all episodes regardless of outcome, since mixed-policy rollouts often yield low success rates. This process repeats until the label budget is exhausted, producing the final dataset $\mathcal{D}_{\text{sup}}$ and the initialized policy π. Notably, BC is a special case of DAgger where the entire budget B is used in a single iteration with the expert policy ($\pi_i = \pi^*$). The complete DAgger procedure is provided in Algorithm 2 (Appendix A.2).

Expert Data Acquisition. To generate expert-labeled data for IL, we deploy the BT in the environment and use a VLM expert to decide whether each condition node is satisfied. To do so, we design detailed, game-specific prompts that describe the game context (e.g., mission, rules) and at each condition-decision step, the input – including the rendered image frame, the game context, and the condition query – is provided to the VLM, which returns a binary decision. These outputs are collected into a dataset of (image, condition query, decision label) tuples. The resulting expert dataset, $\mathcal{D}_{\text{sup}}$, is then used for both policy initialization and expert regularization during RL training. The detailed prompts used for expert data acquisition are provided in Appendix A.6.

5 Experiments

5.1 Experimental Setup

We evaluate our method on seven visual decision-making tasks drawn from three widely used evaluation suites. These include *NumberLine*, *EZPoints*, and *BlackJack* from GymCards (Zhai et al., 2024); *4x4* and *8x8* from FrozenLake (Towers et al., 2024); and *Goto* and *Pickup* from BabyAIText (Paglieri et al., 2025). The detailed descriptions of each task are provided in Appendix A.5. During training, episodes begin from random initial states to ensure exposure to diverse scenarios, and all experiments are executed with four random seeds. For evaluation, we use a fixed set of 100 episodes generated from four seeds not used in training and report averages over these four seeds.

We report four metrics: (1) *Episodic Success Rate*, the proportion of successful episodes; (2) *Decision Accuracy*, the agreement between the learned model's condition-node decisions and those of the expert VLM (GPT-4.1-mini), measured as the proportion of matching decisions on samples from successful expert episodes; (3) *Average Inference Time*, mean runtime per episode; and (4) *Model Size*, the number of parameters in the compact model. Behavior Tree execution uses `py_trees` (Stonier et al., 2025) library. The expert supervision in our experiments is provided by GPT-4.1-mini via API, which performs comparably to or better than open-source multimodal models such as LLaMA-4-Maverick and Qwen3-VL-30B-A3B on public benchmarks. We allocate a budget for each task per random seed: 1,000 VLM queries for *NumberLine* and *FrozenLake 4x4*, and 10,000 VLM queries for *EZPoints*, *BlackJack*, *FrozenLake 8x8*, *BabyAIText Goto*, and *BabyAIText Pickup*. Implementation details, including the network architectures, experimental hardware, and hyperparameters, are provided in Appendix A.4.

We compare several model variants: **IL baselines**, Behavior Cloning and DAgger; **RL base-**

Metrics	Model		GymCards			FrozenLake		BabyAIText		
			NL	BJ	EZP	4x4	8x8	Goto	Pickup	Avg
Success Rate (%) ↑	Expert (GPT-4.1-mini)		100.00	36.75	78.00	98.25	82.00	99.75	99.50	84.89
	IL	BC	**100.00**	40.25	84.50	65.50	13.75	78.75	35.50	59.75
		DA	**100.00**	36.25	44.50	69.75	19.75	66.25	15.75	50.32
	RL	$\mathbf{RL}_{\text{b}}$	**100.00**	28.00	0.00	0.00	0.00	5.25	0.75	19.14
		$\mathbf{RL}_{\text{f}}$ **(ours)**	**100.00**	44.00	0.00	97.75	0.00	86.75	2.50	47.29
	IL + RL (ours)	$\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{1.0}$	**100.00**	42.50	**89.00**	95.50	58.50	95.25	65.00	77.96
		$\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$	**100.00**	44.25	85.00	97.25	64.75	95.50	70.75	**79.64**
	Ablation	$\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$	**100.00**	40.25	0.00	**99.00**	21.75	**98.75**	9.50	52.75
		$\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$	**100.00**	41.75	78.00	95.00	59.25	95.50	**71.00**	77.21
		$\mathbf{init}_{\text{DA}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$	**100.00**	**45.25**	79.00	96.50	**69.25**	93.00	35.25	74.04
Accuracy (%) ↑	Expert (GPT-4.1-mini)		-	-	-	-	-	-	-	-
	IL	BC	**100.00**	91.50	98.90	91.54	81.88	83.98	84.67	90.35
		DA	**100.00**	**93.56**	91.85	91.47	**87.09**	83.03	79.74	89.53
	RL	$\mathbf{RL}_{\text{b}}$	**100.00**	57.84	53.57	77.78	29.13	69.62	53.75	63.10
		$\mathbf{RL}_{\text{f}}$ **(ours)**	95.71	58.52	61.19	73.09	34.66	76.34	73.05	67.51
	IL + RL (ours)	$\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{1.0}$	**100.00**	93.08	**99.21**	**93.45**	82.89	87.07	87.53	**91.89**
		$\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$	**100.00**	85.68	98.62	88.59	78.37	**87.33**	**87.94**	89.50
	Ablation	$\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$	**100.00**	75.95	86.00	87.78	76.09	79.77	78.51	83.44
		$\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$	**100.00**	72.87	95.34	77.44	68.23	84.00	85.49	83.34
		$\mathbf{init}_{\text{DA}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$	**100.00**	88.90	97.04	88.48	86.97	87.15	83.00	90.22

Table 1: Performance comparison of models on **Success Rate** and **Accuracy** across seven tasks. Notations: NL = NumberLine, BJ = BlackJack, EZP = EZPoints; BC = Behavior Cloning, DA = DAgger, $\mathbf{RL}_{\text{b}}$ = baseline RL, $\mathbf{RL}_{\text{f}}$ = factorized RL, ER = Expert Regularization with coefficients 1.0 and 0.1.

lines, the baseline formulation (RL_b), which expands the environment from T steps (episode length in primitive actions) to $K = \sum_{t=1}^{T} n_t$ steps, where n_t is the number of condition nodes traversed at step t, and our factorized formulation (RL_f); **IL+RL**, RL_f with expert regularization with coefficients of 1.0 and 0.1; and **Ablations**, variants that either remove initialization, remove regularization, or use DAgger-based initialization instead of BC initialization.

5.2 Experimental Results

The results are presented in Table 1. Our experiments across the GymCards, FrozenLake, and BabyAIText benchmarks demonstrate that we can successfully learn compact and effective policies that maintain a high level of agreement with the expert. In particular, our integrated CERL framework outperforms both IL and RL baselines individually, while achieving orders-of-magnitude faster inference with a much smaller model size compared to the expert VLM.

CERL Outperforms Baselines. By combining IL and our factorized RL with expert regularization (denoted as $\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$), our approach achieves significantly higher success rates than either IL (BC or DAgger) or RL alone ($\mathbf{RL}_{\text{b}}$ baseline or $\mathbf{RL}_{\text{f}}$ factorized RL). Concretely, the average success rate across all tasks improves to nearly 80% with our IL+RL method, compared to around 60% with IL only and at best roughly 48% with factorized RL alone. Furthermore, our factorized RL ($\mathbf{RL}_{\text{f}}$) consistently outperforms the baseline ($\mathbf{RL}_{\text{b}}$), improving average success rates from approximately 19% to more than 47%. With a regularization coefficient of 0.1, the model achieves the best balance between reward maximization and semantic fidelity, reaching nearly 80% average success and about 90% accuracy. Notably, using a stronger coefficient ($\mathbf{ER}_{1.0}$) yields further gains in faithfulness: $\mathbf{init}_{\text{BC}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{1.0}$ improves average accuracy from 90.35% (BC) to 91.89%, showing that our approach not only boosts success rates but has the potential to enhance the accuracy of imitation learning itself. This confirms the effectiveness of our expert-regularized RL in achieving high success rates while preserving strong semantic faithfulness to expert decisions.

Ablation Study Findings. Our ablation analysis validates the contributions of the individual

components. Ablating the IL-based initialization ($\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$) leads to lower accuracy of about 83.4% as well as lower success rate of about 52.8%. Conversely, IL initialization combined with factorized RL but without expert regularization leads to slightly lower success rate of 77.2% and a more significant decrease in faithfulness. Our proposed framework that consists of IL initialization and expert-regularized RL achieves the best performance across both success rate and accuracy, demonstrating that each component contributes to the overall performance. Additionally, we observe that replacing BC with Dagger ($\mathbf{init}_{\text{DA}}$ + $\mathbf{RL}_{\text{f}}$ w/ $\mathbf{ER}_{0.1}$) tends to underperform compared to BC. This shortfall likely stems from DAgger's limited labeling budget and its mixture policy, which produces fewer successful episodes and thus less effective expert supervision.

Speed, Size, and Cost Efficiency. Our distilled models achieve orders of magnitude faster inference times compared to the expert GPT-4.1-mini VLM, enabling fast execution. Specifically, our IL+RL models infer in approximately 0.09 seconds per episode on average, compared to over 63 seconds for the expert, including API overhead. The compact models are significantly smaller than the expert, with approximately 6.6 million parameters versus approximately 7 billion[2](according to estimates), making them several orders of magnitude lighter. Detailed comparisons of speed and model size across all tasks are provided in Table 3 in the Appendix. Our expert labeling cost for training the models in Table 1 was tightly controlled, limited to about 208,000 expert API calls for condition-node decisions across seven tasks and four seeds, corresponding to a total training cost of about US$96.74.

Experiments with a different expert model. To evaluate the robustness of CERL across different expert models, we conduct additional experiments using Qwen3.5-Flash (released February 2026) as the expert, while keeping the same hyperparameters as those used with GPT-4.1-mini. All results are averaged over four random seeds. CERL achieves comparable performance under both GPT-4.1-mini and Qwen3.5-Flash supervision. In particular, the average success rates remain close across experts (e.g., 79.64% vs. 79.14% for $\mathbf{ER}_{0.1}$), while accuracy is also close (89.50% vs. 88.08%).

[2]Unofficial estimate, e.g., https://amigochat.io/gpt-4-1-mini

	GPT-4.1-mini	Qwen3-30B		Qwen3.5-Flash	
	Def.	Def.	Rpt x3	Def.	Rpt x3
NL	100.00	100.00	100.00	100.00	100.00
BJ	36.75	31.75	34.50	39.00	37.75
EZP	78.00	3.25	57.75	46.00	70.25
FL4x4	98.25	98.25	100.00	97.25	99.75
FL8x8	82.00	73.25	85.50	89.75	90.74
Goto	99.75	100.00	100.00	100.00	99.00
Pickup	99.50	99.75	100.00	99.50	99.75
Avg	84.89	72.32	82.54	81.64	85.32

Table 2: Expert success rate (%) comparison across VLMs. GPT-4.1-mini is evaluated with default prompting only, while Qwen3-30B (Qwen3-VL-30B-A3B-Instruct) and Qwen3.5-Flash are evaluated with both default prompting and prompt repetition (x3).

Notably, even when the expert's success rate differs significantly across tasks (e.g., EZP: 78.00 vs. 46.00), CERL trained with Qwen3.5-Flash is able to recover strong performance (99.50% success with $\mathbf{ER}_{1.0}$), demonstrating robustness to weaker or noisier supervision. These results indicate that CERL is not tightly coupled to a specific expert model and generalizes well across different VLM supervisors. Detailed experimental results are provided in Table 4 in the Appendix.

Noisy Expert. VLM experts may occasionally produce incorrect or biased labels, which CERL could inherit. However, techniques such as self-consistency (Wang et al., 2023), self-refinement (Madaan et al., 2023), and prompt repetition (Leviathan et al., 2025) can improve expert accuracy. To demonstrate this, we evaluate Qwen3-VL-30B-A3B-Instruct and Qwen3.5-Flash (released February 2026) under default prompting as well as prompt repetition. As shown in Table 2, prompt repetition (x3) consistently improves average success rates, from 72.32% to 82.54% for Qwen3-VL-30B-A3B-Instruct and from 81.64% to 85.32% for Qwen3.5-Flash. The corresponding API costs are approximately US$349.39 and US$31.45, respectively, with all results averaged over four random seeds.

Qualitative Analysis. We observe several interesting cases where CERL outperforms the expert model. Specifically, in BlackJack, we observe cases where CERL achieves higher success rates than the expert, suggesting that the expert BT may follow a suboptimal strategy in certain states. Concretely, under the current BlackJack BT, the expert tends to "hit" when the sum is below 17, which

can sometimes lead to busting (i.e., exceeding 21). In contrast, CERL learns to "stay" in states where the risk of busting is high (e.g., when the sum is close to 17 and the remaining deck distribution is unfavorable), resulting in higher win rates.

In EZPoints, we observe that the expert (GPT-4.1-mini) exhibits difficulty in evaluating arithmetic visual conditions such as "Is '+' a better choice based on the remaining values?". RL optimization corrects this limitation, enabling the distilled model to surpass the expert. Meanwhile, expert regularization preserves semantic alignment on the remaining condition decisions, maintaining approximately 99% agreement with the expert's decisions.

6 Related Work

A large body of research has combined IL with RL to improve policy performance, e.g., (Xie et al., 2021; Xue et al., 2023; Luo et al., 2024). However, they typically do not emphasize maintaining semantic fidelity to expert behavior or integrating interpretability mechanisms. In contrast, our work leverages IL to preserve faithfulness to expert decisions while using RL for reward optimization, all within an interpretable framework based on BTs.

Although RL has been applied to BTs to improve adaptability and performance, typically by optimizing local node-level decisions such as adjusting fallback priorities (Xu et al., 2022), embedding learned policies into action nodes (Li et al., 2024), or constructing BT structures through RL-based synthesis (Huang et al., 2025), these approaches are largely limited to symbolic or low-dimensional feature spaces as RL inputs. This limitation restricts their applicability to high-dimensional perceptual inputs such as images and language without costly feature engineering. In addition, many of these methods modify or replace standard BT nodes, for example, by embedding opaque neural policies into action nodes or altering control-flow logic, thereby weakening alignment with the human designer's intent and reducing the transparency and interpretability that BTs are meant to provide. In contrast, our approach preserves the standard BT structure by keeping action and control-flow nodes unchanged, while training only condition nodes with lightweight neural models regularized by expert supervision to retain their intended semantics.

Decision trees (Breiman et al., 2017) have also been explored as interpretable policy representations in RL. Prior studies primarily fall into two directions. Distillation-based methods extract tree policies from trained RL agents to enhance transparency and verifiability (Bastani et al., 2018; Acero and Li, 2024), while differentiable methods learn tree parameters end-to-end through gradient-based optimization (Silva et al., 2020; Wen et al., 2025). To our knowledge, no prior approach has introduced a factorized policy formulation that enables efficient root-to-leaf sampling-based policy-gradient optimization for differentiable tree structures, nor has it enabled direct training of expert-designed, tree-structured policies from high-dimensional perceptual inputs for sequential decision-making.

7 Conclusion

We presented a unified framework for interpretable sequential decision-making based on expert-designed behavior trees that operate on high-dimensional perceptual inputs, integrating imitation learning and reinforcement learning through expert regularization. Our approach uses a limited set of expert labels, leveraging them both for initialization and as a continuous regularizer to preserve semantic alignment with expert decisions. Reinforcement learning drives performance improvement, while expert regularization constrains the policy to remain close to expert behavior. The RL component is built on factorized node-level distributions that aggregate condition-node decisions into a single unit, simplifying credit assignment under sparse rewards. Across the GymCards, FrozenLake, and BabyAIText suites, we showed that our approach achieves strong success rates, maintains semantic consistency with expert decisions.

These results highlight that natural language serves as an effective semantic interface between symbolic logic and visual perception, enabling compositional reasoning that is otherwise difficult to encode as symbolic rules or to learn from visual features alone. Overall, our results demonstrate that CERL effectively leverages language-based condition queries to evaluate complex spatial and relational conditions within an interpretable symbolic structure, removing the need for large VLMs at inference time and thereby enabling real-time applications where high-latency VLMs are impractical.

Limitations

We highlight two limitations that are required for the success of our approach. Firstly, our framework is intended for an expert-designed Behavior Tree (BT) that encodes an effective policy for the task. CERL does not learn the BT structure or the condition queries, but only learns lightweight condition checkers. Therefore, if a sub-optimal BT is provided, the learned agent may inherit structural limitations from this design. Secondly, our method requires an accurate expert model (e.g., a VLM) capable of reliably evaluating natural-language condition queries over perceptual inputs to provide labels for the imitation learning. When the labels are noisy or incorrect, performance can degrade. However, as shown in Table 2, techniques like prompt repetition can be utilized to significantly improve supervision quality. Extending our framework to address these limitations, e.g., via automated BT synthesis or adaptation, or learning under weaker and noisy supervision, are promising directions for future work.

Acknowledgments

This research was supported by grant number DSICGY3R1P18 from the Data Sciences Institute at the University of Toronto. The authors also gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Fernando Acero and Zhibin Li. 2024. Distilling reinforcement learning policies for interpretable robot locomotion: Gradient boosting machines and symbolic regression. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6840–6847. IEEE.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. 2018. Verifiable reinforcement learning via policy extraction. *Advances in Neural Information Processing Systems*, 31:2499–2509.

Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. Robot programming by demonstration. In *Springer Handbook of Robotics*, pages 1371–1394. Springer.

Leo Breiman, Jerome Friedman, Richard A Olshen, and Charles J Stone. 2017. *Classification and regression trees*. Chapman and Hall/CRC.

Michele Colledanchise and Petter Ögren. 2018. *Behavior trees in robotics and AI: an introduction*. CRC Press.

Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. 2018. Learning of behavior trees for autonomous agents. *IEEE Transactions on Games*, 11(2):183–189.

Akash Ghosh, Arkadeep Acharya, Sriparna Saha, Vinija Jain, and Aman Chadha. 2024. Exploring the frontier of vision-language models: a survey of current methodologies and future directions. *arXiv preprint arXiv:2404.07214*.

Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S Brown, and Ken Goldberg. 2022. ThriftyDAgger: budget-aware novelty and risk gating for interactive imitation learning. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, pages 598–608. PMLR.

Hao Hu, Xiaoliang Jia, Kuo Liu, and Bingyang Sun. 2021. Self-adaptive traffic control model with behavior trees and reinforcement learning for AGV in industry 4.0. *IEEE Transactions on Industrial Informatics*, 17(12):7968–7979.

Yu Huang, Ziji Wu, Kexin Ma, and Ji Wang. 2025. Differentiable synthesis of behavior tree architectures and execution nodes. In *Proceedings of the 2nd International Conference on Neuro-Symbolic Systems (NeuS 2025)*, volume 288. PMLR.

Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. 2022. A survey of behavior trees in robotics and AI. *Robotics and Autonomous Systems*, 154:104096.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. 2019. HG-DAgger: interactive imitation learning with human experts. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2025. Prompt repetition improves non-reasoning llms. *arXiv preprint arXiv:2512.14982*.

Xianglong Li, Yuan Li, Jieyuan Zhang, Xinhai Xu, and Donghong Liu. 2024. Embedding multi-agent reinforcement learning into behavior trees with unexpected interruptions. *Complex & Intelligent Systems*, 10(3):3273–3282.

Jianlan Luo, Perry Dong, Yuexiang Zhai, Yi Ma, and Sergey Levine. 2024. RLIF: interactive imitation learning as reinforcement learning. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Ryan Marcotte and Howard J Hamilton. 2017. Behavior trees for modelling artificial intelligence in games: a tutorial. *The Computer Games Journal*, 6(3):171–184.

W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. 2019. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080.

Petter Ögren and Christopher I Sprague. 2022. Behavior trees in robot control systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1):81–107.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, and 1 others. 2025. BALROG: benchmarking agentic LLM and VLM reasoning on games. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 627–635. JMLR Workshop and Conference Proceedings.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Andrew Silva, Taylor Killian, Ivan Jimenez, Sung-Hyun Son, and Matthew Gombolay. 2020. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1855–1865. PMLR.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471.

Daniel Stonier and 1 others. 2025. py_trees: Pythonic behaviour trees for python. https://github.com/splintered-reality/py_trees.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12:1057–1063.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulao, Andreas Kallinteris, Markus Krimmel, Arjun KG, and 1 others. 2024. Gymnasium: a standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*.

Naoki Wake, Atsushi Kanehira, Jun Takamatsu, Kazuhiro Sasabuchi, and Katsushi Ikeuchi. 2025. VLM-driven behavior tree for context-aware task planning. *arXiv preprint arXiv:2501.03968*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*.

Yongyan Wen, Siyuan Li, Rongchang Zuo, Lei Yuan, Hangyu Mao, and Peng Liu. 2025. Skilltree: Explainable skill-based deep reinforcement learning for long-horizon control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 21491–21500.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. 2021. Policy finetuning: bridging sample-efficient offline and online reinforcement learning. *Advances in Neural Information Processing Systems*, 34:27395–27407.

Jiahua Xu, Yunhan Lin, Haotian Zhou, and Huasong Min. 2022. Generating manipulation sequences using reinforcement learning and behavior trees for peg-in-hole task. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2715–2720. IEEE.

Zhenghai Xue, Zhenghao Peng, Quanyi Li, Zhihan Liu, and Bolei Zhou. 2023. Guarded policy optimization with imperfect online demonstrations. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Maryam Zare, Parham M Kebria, Abbas Khosravi, and Saeid Nahavandi. 2024. A survey of imitation learning: algorithms, recent developments, and challenges. *IEEE Transactions on Cybernetics*.

Simon Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Peter Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and 1 others. 2024. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *Advances in Neural Information Processing Systems*, 37:110935–110971.

A Appendix

A.1 Proofs

This section provides detailed proofs for the key theoretical results presented in the main paper.

Proposition 1. (Distribution over Decision Sequences Induces a Valid Action Policy) *Let $p(\mathbf{y}_t \mid s_t)$ be a normalized probability distribution over decision sequences (i.e., $\sum_{\mathbf{y}_t} p(\mathbf{y}_t \mid s_t) = 1$). Assume that the BT $\mathcal{T}$ is deterministic, and for every sequence $\mathbf{y}_t$ with $p(\mathbf{y}_t \mid s_t) > 0$, the BT returns exactly one primitive action $a_t = \mathcal{T}(\mathbf{y}_t) \in \mathcal{A}$. Define the induced action policy*

$$\pi(a_t \mid s_t) \;=\; \sum_{\mathbf{y}_t:\, \mathcal{T}(\mathbf{y}_t)=a_t} p(\mathbf{y}_t \mid s_t). \tag{8}$$

Then $\pi(a_t \mid s_t)$ is a valid probability distribution over $\mathcal{A}$.

Proof. Each term $p(\mathbf{y}_t \mid s_t)$ is nonnegative, so $\pi(a_t \mid s_t) \geq 0$ for every $a_t \in \mathcal{A}$. To show the probabilities over actions sum to 1, add them up:

$$\sum_{a_t \in \mathcal{A}} \pi(a_t \mid s_t) = \sum_{a_t \in \mathcal{A}} \sum_{\mathbf{y}_t:\, \mathcal{T}(\mathbf{y}_t)=a_t} p(\mathbf{y}_t \mid s_t). \tag{9}$$

Because the BT is deterministic, each sequence that can occur (i.e., has $p(\mathbf{y}_t \mid s_t) > 0$) leads to exactly one action, so each sequence $\mathbf{y}_t$ appears in exactly one of the inner sums. Sequences with zero probability do not contribute. Therefore the double sum counts each occurring sequence exactly once, giving

$$\sum_{a_t \in \mathcal{A}} \pi(a_t \mid s_t) = \sum_{\mathbf{y}_t} p(\mathbf{y}_t \mid s_t) = 1. \tag{10}$$

Thus $\pi(a_t \mid s_t)$ is nonnegative and sums to 1, so it is a valid probability distribution over $\mathcal{A}$. □

Proposition 2. (Expectation preservation under deterministic mapping) *Let $a_t = \mathcal{T}(\mathbf{y}_t)$ for a deterministic mapping $\mathcal{T}$, and let $p(\mathbf{y}_t \mid s_t)$ be a valid distribution over decision sequences. Then, for any function f over actions (e.g., reward or advantage functions),*

$$\mathbb{E}_{a_t \sim \pi(\cdot \mid s_t)}[f(a_t)] = \mathbb{E}_{\mathbf{y}_t \sim p(\cdot \mid s_t)}[f(\mathcal{T}(\mathbf{y}_t))]. \tag{11}$$

Proof. By the law of total probability (Proposition 1, Eq. (8)),

$$\begin{aligned}
&\mathbb{E}_{a_t \sim \pi(\cdot \mid s_t)}[f(a_t)] && (12)\\
&= \sum_{a_t} \pi(a_t \mid s_t) f(a_t) && (13)\\
&= \sum_{a_t} \left(\sum_{\mathbf{y}_t:\, \mathcal{T}(\mathbf{y}_t)=a_t} p(\mathbf{y}_t \mid s_t) \right) f(a_t) && (14)\\
&= \sum_{\mathbf{y}_t} p(\mathbf{y}_t \mid s_t) f(\mathcal{T}(\mathbf{y}_t)) && (15)\\
&= \mathbb{E}_{\mathbf{y}_t \sim p(\cdot \mid s_t)}[f(\mathcal{T}(\mathbf{y}_t))]. && (16)
\end{aligned}$$

The crucial step is the move from Eq. (14) to Eq. (15). Since each sequence $\mathbf{y}_t$ deterministically maps to exactly one action $a_t = \mathcal{T}(\mathbf{y}_t)$, Eq. (14) becomes equivalent to summing once over all sequences and evaluating f at the induced action $\mathcal{T}(\mathbf{y}_t)$. □

Theorem 1. (Policy gradient for BT-driven policies) *Let p denote the distribution over BT decision sequences, which induces the valid action policy π as established in Proposition 1. Using the expectation-preservation property of Proposition 2, the policy gradient, $\nabla_\theta J(\theta)$, is*

$$\mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \sum_{i=1}^{n_t} A_t \, \nabla_\theta \log q_\theta(y_{t,i} \mid s_t, q_i) \right], \tag{17}$$

where A_t is any valid environment-time advantage, and the expectation is over trajectories τ sampled from the environment distribution induced by π.

Proof. We define the RL environment as a Markov Decision Process, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Let $\tau = (s_0, \mathbf{y}_0, a_0, r_0, s_1, \ldots, s_{T+1})$ denote a trajectory. Its probability under the induced action policy, $\pi(\tau)$, is given by

$$\begin{aligned}
& p_0(s_0) \prod_{t=0}^{T} \left[\pi(a_t \mid s_t)\; P(s_{t+1} \mid s_t, a_t) \right] && (18)\\
=\; & p_0(s_0) \prod_{t=0}^{T} \left[p(\mathbf{y}_t \mid s_t)\; P(s_{t+1} \mid s_t, a_t) \right] && (19)\\
=\; & p_0(s_0) \prod_{t=0}^{T} \left[\prod_{i=1}^{n_t} q_\theta(y_{t,i} \mid s_t, q_i)\; P(s_{t+1} \mid s_t, a_t) \right], && (20)
\end{aligned}$$

where $p_0(\cdot)$ is the initial state distribution.

By the log-derivative trick (Sutton et al., 1999; Williams, 1992), for any differentiable policy, $\nabla_\theta J(\theta)$ is

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R(\tau)] = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\,\nabla_\theta \log \pi(\tau)\right]. \tag{21}$$

The full trajectory log-probability is

$$\log \pi(\tau) = \log p_0(s_0) + \sum_{t=0}^{T}\sum_{i=1}^{n_t} \log q_\theta(y_{t,i} \mid s_t, q_i) + \sum_{t=0}^{T} \log P(s_{t+1} \mid s_t, a_t). \tag{22}$$

The initial state distribution p_0 and the transition probability distribution P do not depend on θ, so their gradient vanishes. Thus,

$$\nabla_\theta \log \pi(\tau) = \sum_{t=0}^{T}\sum_{i=1}^{n_t} \nabla_\theta \log q_\theta(y_{t,i} \mid s_t, q_i), \tag{23}$$

substituting $\nabla_\theta \log \pi(\tau)$ into the expectation, and using that the return $R(\tau)$ can be replaced by an advantage-weighted sum without changing the expectation, we obtain $\nabla_\theta J(\theta)$,

$$\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\sum_{i=1}^{n_t} A_t\,\nabla_\theta \log q_\theta\big(y_{t,i} \mid s_t, q_i\big)\right]. \tag{24}$$

□

In practice, we optimize a surrogate objective using Proximal Policy Optimization (Schulman et al., 2017), which maximizes the following objective, $J_{\text{PPO}}(\theta)$:

$$\mathbb{E}_t\left[\min\left(\rho_t(\theta)A_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right], \tag{25}$$

where the importance sampling ratio is defined as

$$\rho_t(\theta) := \frac{\prod_{i=1}^{n_t} q_\theta(y_{t,i} \mid s_t, q_i)}{\prod_{i=1}^{n_t} q_{\theta_{\text{old}}}(y_{t,i} \mid s_t, q_i)}. \tag{26}$$

The advantage estimator A_t is computed using Generalized Advantage Estimation (Schulman et al., 2015). The full PPO loss combines the clipped surrogate in Eq. (25) with the importance sampling ratio defined over decision sequences (Eq. (26)), plus a value loss based on the state s_t and an entropy bonus computed over the node-level outputs $q_\theta(y_{t,i} \mid s_t, q_i)$, following the standard PPO (Schulman et al., 2017).

A.2 Algorithms

This section presents the complete pseudocode for the algorithms discussed in the main paper, including CERL (Condition-node Expert-Regularized Reinforcement Learning) and DAgger with Fixed Label Budget. These implementations correspond to Algorithms 1 and 2 referenced in the main text.

A.3 Extended and Additional Experimental Results

Speed and Model Size. As shown in Table 3, our compact models demonstrate significant advantages in both inference speed and size compared to the expert GPT-4.1-mini VLM. Across all tasks, the distilled IL+RL variants achieve average inference times of roughly 0.09 seconds per episode, while the expert requires more than 63 seconds on average, including API overhead. This efficiency gain corresponds to a speedup of over three orders of magnitude. In terms of size, each compact model contains approximately 6.6 million parameters, compared to the expert's estimated 7 billion, making our approach both computationally and memory efficient.

Effect of Expert Data Size. To evaluate the impact of expert dataset size, we train models using behavior cloning with $0.5 \times 10{,}000$ and $0.2 \times 10{,}000$ expert labels on the *FrozenLake 8x8* and *BabyAIText Pickup* tasks, averaging results over four random seeds. As shown in Figure 3, reducing the amount of expert-labeled data leads to a clear decline in both success rate and accuracy.

Effect of Noisy Expert Labels. To evaluate the impact of noisy expert supervision, we train models with synthetic label noise by randomly flipping expert labels with probabilities of 0%, 10%, and 25%, averaging results over four random seeds. As shown in Figure 4, increasing the noise level in the expert labels leads to a clear degradation in both success rate and accuracy.

Effect of Traversal Depth. We analyze the scalability of our method with respect to traversal depth by artificially extending the behavior tree (BT) in *BabyAIText Goto*. Specifically, each action node is replaced with `Selector(Sequence(condition_placeholder, action), action)`, increasing the number of condition nodes from 7 to 14. All experiments follow the same hyperparameters in Table 8 and are conducted with a single random seed. We train both

Algorithm 1 CERL: Condition-node Expert-regularized Reinforcement Learning

Input: Behavior Tree $\mathcal{T}$, initial policy π (parameters θ), environment $\mathcal{E}$, expert dataset $\mathcal{D}_{\text{sup}}$, expert-regularization (ER) coefficient λ_{ER}
Output: policy π

Initialize π on $\mathcal{D}_{\text{sup}}$ via imitation learning (e.g., Behavior Cloning or DAgger - Algorithm 2)
for each iteration **do**
  Collect trajectories $\mathcal{D}$ in $\mathcal{E}$ using BT $\mathcal{T}$ and policy π
  Compute returns and advantages (GAE) from $\mathcal{D}$
  for each PPO epoch **do**
    for each minibatch $(\mathbf{S}, \mathbf{Y}, \mathbf{A}, R, \hat{A})$ in $\mathcal{D}$ **do**
      Compute L_{PPO}: PPO loss based on importance ratio Eq. (26), including value and entropy
      Sample minibatch $(\mathbf{S}_{\text{sup}}, \mathbf{Q}_{\text{sup}}, \mathbf{Y}_{\text{sup}})$ from $\mathcal{D}_{\text{sup}}$
      Compute L_{ER}: cross-entropy loss between $\pi(y_{\text{sup}} \mid s_{\text{sup}}, q_{\text{sup}})$ and expert labels y_{sup}
      Update θ to minimize $L_{\text{PPO}} + \lambda_{\text{ER}} L_{\text{ER}}$
return π_θ

Algorithm 2 DAgger with Fixed Label Budget

Input: Behavior Tree $\mathcal{T}$, environment $\mathcal{E}$, expert policy π^*, initial policy π (parameters θ), total budget B, per-iteration quota q, initial mix β_0, decay $\alpha \in (0, 1)$
Output: Aggregated dataset $\mathcal{D}_{\text{sup}}$, policy π
Notation: $x = (\text{image}, \text{condition query})$ for learner input;
$x^+ = (\text{image}, \text{game context} + \text{condition query})$ for expert input

$\hat{\pi}_0 \leftarrow \pi$; $\mathcal{D}_{\text{sup}} \leftarrow \emptyset$; $b \leftarrow B$; $i \leftarrow 0$
while $b > 0$ **do**
  $\beta_i \leftarrow \min\{1, \beta_0 \alpha^i\}$; $c \leftarrow \min\{q, b\}$
  Define rollout mixture: $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
  while $c > 0$ **do**
    Roll out $\mathcal{T}$ for one tick in $\mathcal{E}$ using π_i
    Get $D_i = \{(x, \pi^*(x^+))\}$ of condition queries visited by π_i
    $\mathcal{D}_{\text{sup}} \leftarrow \mathcal{D}_{\text{sup}} \cup D_i$; $c \leftarrow c - |D_i|$; $b \leftarrow b - |D_i|$
  Update $\hat{\pi}_i$ on $\mathcal{D}_{\text{sup}}$ with cross-entropy
  $i \leftarrow i + 1$
$\pi \leftarrow \hat{\pi}_i$
return $(\mathcal{D}_{\text{sup}}, \pi)$

RL baselines on the original and extended BTs for 1 million steps, and additionally extend training to 1.5 million steps for the deeper BT to examine convergence. As shown in Table 5, our factorized $\mathbf{RL}_{\text{f}}$ remains robust under increased traversal depth, achieving near-complete recovery of performance with additional training (1.5M steps), whereas the baseline formulation, $\mathbf{RL}_{\text{b}}$, fails to improve even under default-length training. This indicates that our factorized formulation scales effectively with deeper trees, maintaining both success rate and decision accuracy despite increased condition evaluations per tick.

A.4 Experimental Details

A.4.1 Implementation Details.

Reward formulation. Our RL agent optimizes the environment reward without additional shaping. Reward normalization is disabled. Details of the reward functions for each environment are provided in Appendix A.5.

Termination Criteria. All tasks use a fixed maximum episode length, with a default of 100 steps. Episodes terminate when the environment reaches a terminal state (task completion or failure) or is truncated (e.g., due to the maximum step limit). Details of the termination conditions for each envi-

Metrics	Model		GymCards			FrozenLake		BabyAIText		
			NL	BJ	EZP	4x4	8x8	Goto	Pickup	Avg
Inference Time (s/ep) ↓	Expert (GPT-4.1-mini)		4.82	13.24	39.58	60.18	241.51	31.86	52.54	63.39
	IL	BC	**0.01**	0.17	0.08	0.03	**0.06**	0.09	0.37	0.12
		DA	**0.01**	0.17	**0.07**	0.05	0.28	0.14	0.46	0.17
	RL	$\mathbf{RL_b}$	**0.01**	**0.15**	0.05	0.28	1.53	0.42	0.35	0.40
		$\mathbf{RL_f}$ (ours)	**0.01**	0.16	0.14	**0.02**	0.72	0.09	0.35	0.21
	IL + RL (ours)	$\mathbf{init_{BC}}$ + $\mathbf{RL_f}$ w/ $\mathbf{ER_{1.0}}$	**0.01**	0.17	0.11	0.04	0.11	0.07	0.16	**0.09**
		$\mathbf{init_{BC}}$ + $\mathbf{RL_f}$ w/ $\mathbf{ER_{0.1}}$	**0.01**	0.17	0.11	0.03	0.14	0.06	**0.11**	**0.09**
	Ablation	$\mathbf{RL_f}$ w/ $\mathbf{ER_{0.1}}$	**0.01**	**0.15**	0.08	**0.02**	0.11	**0.03**	0.42	0.12
		$\mathbf{init_{BC}}$ + $\mathbf{RL_f}$	**0.01**	0.17	0.12	0.04	0.19	0.06	0.16	0.11
		$\mathbf{init_{DA}}$ + $\mathbf{RL_f}$ w/ $\mathbf{ER_{0.1}}$	**0.01**	0.16	0.12	0.03	0.36	0.11	0.43	0.18
Model Size (# params) ↓	Expert (GPT-4.1-mini)		∼7B[3]							
	Others		6.6M							

Table 3: Inference time and model size across seven tasks. NL = NumberLine, BJ = BlackJack, EZP = EZPoints; BC = Behavior Cloning, DA = DAgger, $\mathbf{RL_b}$ = baseline RL, $\mathbf{RL_f}$ = factorized RL, ER = Expert Regularization (coefficients 1.0 and 0.1). Bold values indicate the fastest among models with non-zero success rate.

Metrics	Model		GymCards			FrozenLake		BabyAIText		
			NL	BJ	EZP	4x4	8x8	Goto	Pickup	Avg
Success Rate (%) ↑	Expert (GPT-4.1-mini)		**100.00**	36.75	**78.00**	**98.25**	82.00	99.75	**99.50**	**84.89**
	Expert (Qwen3.5-Flash)		**100.00**	**39.00**	46.00	97.25	**89.75**	**100.00**	**99.50**	81.64
	$\mathbf{init_{BC}}$ + $\mathbf{RL_f}$ w/ $\mathbf{ER_{1.0}}$	GPT-4.1-mini	100.00	**42.50**	89.00	95.50	**58.50**	**95.25**	**65.00**	**77.96**
		Qwen3.5-Flash	100.00	38.00	**99.50**	**96.75**	51.00	94.75	60.50	77.21
	$\mathbf{init_{BC}}$ + $\mathbf{RL_f}$ w/ $\mathbf{ER_{0.1}}$	GPT-4.1-mini	100.00	**44.25**	85.00	97.25	64.75	**95.50**	**70.75**	**79.64**
		Qwen3.5-Flash	100.00	42.50	**87.25**	**98.25**	**70.50**	94.50	61.00	79.14
Accuracy (%) ↑	Expert (GPT-4.1-mini)		-	-	-	-	-	-	-	-
	Expert (Qwen3.5-Flash)		-	-	-	-	-	-	-	-
	$\mathbf{init_{BC}}$ + $\mathbf{RL_f}$ w/ $\mathbf{ER_{1.0}}$	GPT-4.1-mini	100.00	**93.08**	**99.21**	**93.45**	**82.89**	**87.07**	**87.53**	**91.89**
		Qwen3.5-Flash	100.00	90.39	98.10	89.36	82.45	84.79	83.79	89.84
	$\mathbf{init_{BC}}$ + $\mathbf{RL_f}$ w/ $\mathbf{ER_{0.1}}$	GPT-4.1-mini	100.00	85.68	**98.62**	**88.59**	78.37	**87.33**	**87.94**	**89.50**
		Qwen3.5-Flash	100.00	**87.09**	96.05	83.83	**79.65**	83.48	86.48	88.08

Table 4: Comparison of Expert and CERL (IL+RL) performance using GPT-4.1-mini and Qwen3.5-Flash. Bold indicates the better result between GPT-4.1-mini and Qwen3.5-Flash.

Model	Metric	Default (1M)	Ext. (1M)	Ext. (1.5M)
$\mathbf{RL_b}$	Success (%)	55.56 / 6.00	55.56 / 6.00	56.25 / 6.00
	Accuracy (%)	– / 71.26	– / 71.34	– / 73.85
	#Cond./tick	– / 2.00	– / 3.00	– / 3.00
$\mathbf{RL_f}$	Success (%)	99.47 / 99.00	97.30 / 74.00	100.00 / 99.00
	Accuracy (%)	– / 80.95	– / 79.74	– / 81.99
	#Cond./tick	– / 2.25	– / 3.20	– / 3.38

Table 5: Traversal depth ablation on *BabyAIText Goto*. Values are reported as Train / Eval.

ronment are provided in Appendix A.5.

Input and Output Representation. The compact model receives two inputs: (i) rendered RGB observations resized to 128×128 pixels using bilinear interpolation, and (ii) a discrete condition-query ID representing the queried condition node. Pixel values are converted from $[0, 255]$ to $[0, 1]$, and then normalized channel-wise with mean $(0.5, 0.5, 0.5)$ and standard deviation $(0.5, 0.5, 0.5)$, effectively mapping inputs to the range $[-1, 1]$ (e.g., a pixel value of 0 maps to -1, 0.5 to 0, and 1 to 1, using $(x - 0.5)/0.5$).

The condition-query ID indexes the set of unique natural-language condition queries in the behavior tree, and is mapped to a learned embedding. The model combines the visual representation and the condition-query embedding, and outputs logits for a binary True/False decision via a multi-layer perceptron (MLP). The detailed model architecture is provided in Table 6.

The VLM expert, in contrast, receives the rendered image together with task-specific textual con-

[3]Unofficial estimate, e.g., https://amigochat.io/gpt-4-1-mini

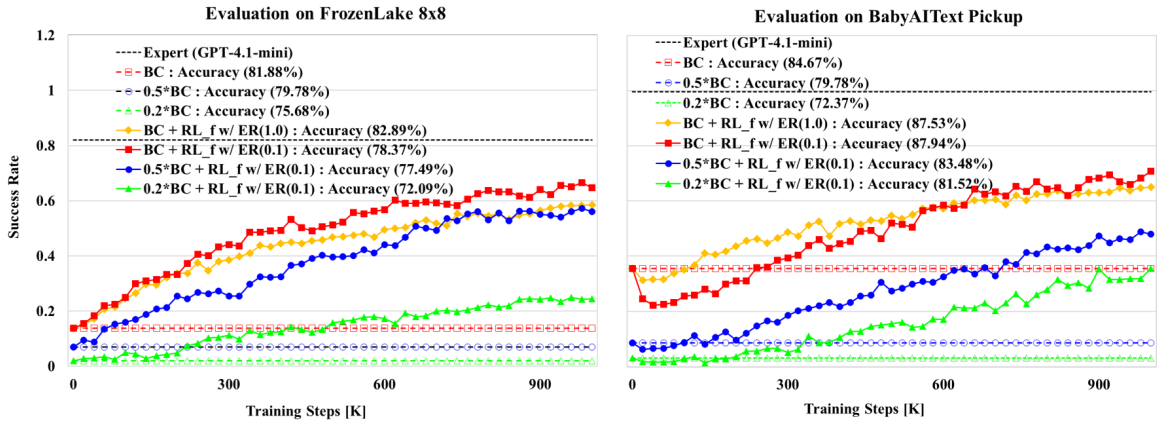


Figure 3: Effect of expert dataset size on *FrozenLake 8x8* (left) and *BabyAIText Pickup* (right). While expert-regularized RL improves performance even with limited expert labels, models trained with fewer labels (0.5× and 0.2× of the full dataset) show progressively lower success rates and reduced agreement with expert decisions (accuracy), underscoring the need for minimal expert supervision for overall performance. Results are averaged over four random seeds.

text and returns a JSON object containing a rationale and a binary True/False decision. The detailed prompts for the textual context are provided in Appendix A.6.

Handling Malformed or Unstable Expert Outputs. We do not use calibrated uncertainty estimation or an explicit confidence score for expert labels. Instead, robustness is handled procedurally. Temporary API failures (e.g., connection errors, timeouts, or rate limits) are handled by automatically retrying the request after progressively longer waiting intervals. The returned text is then post-processed to recover a binary decision whenever possible. Concretely, the parser first attempts to extract a valid JSON object from the response. If this fails, it falls back to scanning the raw text for an explicit `True` or `False` label and uses that recovered label as the binary decision. If neither form can be recovered, the response is treated as invalid and mapped to a negative (`False`) label. The implementation also supports optional self-consistency voting and prompt repetition for added robustness, though both are disabled by default in our main experiments.

A.4.2 Model Architectures.

We implemented two variants of the actor–critic network for condition-node policies: our proposed factorized RL ($\mathbf{RL_f}$) and the baseline RL ($\mathbf{RL_b}$). Both share the same image and text encoder architecture but differ in how the critic's state representation is defined. The detailed architectures are provided in Table 6 and Table 7.

In the factorized formulation ($\mathbf{RL}_\text{f}$), the environment state at step t is defined as $s_t = \text{image}_t$. Each condition-node query (e.g., a sentence) is mapped to a token in the vocabulary, with identical queries sharing the same token id and embedding. The text encoder retrieves a single embedding vector for the query token, which is combined with the image feature by the actor to produce condition decisions. The critic, however, depends only on the image feature and computes $V(s_t)$, reducing the critic's input domain.

In the baseline setting ($\mathbf{RL}_\text{b}$), each condition decision (t, i) is treated as a separate state $s_{t,i} = (\text{image}_t, \text{text}_{t,i})$, where $\text{text}_{t,i}$ is the query associated with condition node i. Queries are again mapped to vocabulary tokens, with identical queries sharing the same token id and embedding. The text encoder retrieves the query embedding, which the actor combines with the image feature to produce condition decisions. Unlike $\mathbf{RL}_\text{f}$, the critic also takes the query into account, computing $V(s_{t,i})$ from the joint image–text input. This enlarges the critic's input domain and complicates credit assignment under sparse rewards.

A.4.3 Computing Environment.

Experiments were conducted on two Windows 11 workstations. The first machine is equipped with an Intel Core i9-14900F CPU @ 2.00 GHz, 32 GB RAM, and an NVIDIA GeForce RTX 4090 GPU

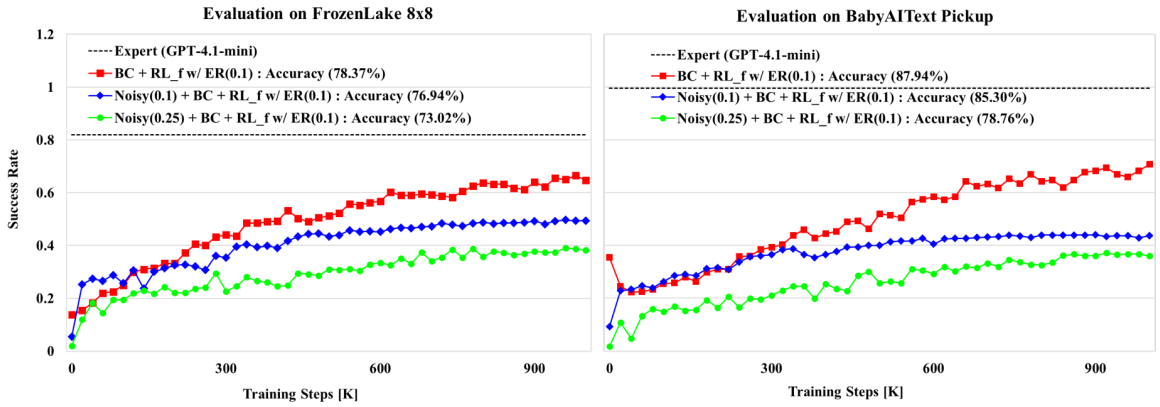


Figure 4: Effect of noisy expert supervision on *FrozenLake 8x8* (left) and *BabyAIText Pickup* (right). Models trained with noisy labels (10% and 25%) show progressively lower success rates and reduced agreement with expert decisions (accuracy), underscoring the importance of accurate expert supervision for robust performance. Results are averaged over four random seeds.

Component	**Layer**	**Output dim.**
Image encoder	Conv2d(3, 64, 3×3, 3) + ReLU Conv2d(64, 64, 3×3, 1) + ReLU Flatten + Linear(d_{flatten}, 64)	d_{img}=64
Text encoder	Embedding ($\|token\|$, $d_{\text{emb}} = 32$) → Linear(32, 32)	d_{text}=32
Actor	Linear($d_{\text{img}} + d_{\text{text}}$, 64) + ReLU → Linear(64, 2)	logits (Success/Failure)=2
Critic	Linear(d_{img}, 64) + ReLU → Linear(64, 1)	$V(s_t)$=1
Initialization	Orthogonal init (all linear + embeddings)	–

Table 6: $\mathbf{RL}_{\text{f}}$ (ours) network architecture.

Component	**Layer**	**Output dim.**
Image encoder	Conv2d(3, 64, 3×3, 3) + ReLU Conv2d(64, 64, 3×3, 1) + ReLU Flatten + Linear(d_{flatten}, 64)	d_{img}=64
Text encoder	Embedding ($\|token\|$, d_{emb}=32) → Linear(32, 32)	d_{text}=32
Actor	Linear($d_{\text{img}} + d_{\text{text}}$, 64) + ReLU → Linear(64, 2)	logits (Success/Failure)=2
Critic	Linear($d_{\text{img}} + d_{\text{text}}$, 64) + ReLU → Linear(64, 1)	$V(s_{t,i})$=1
Initialization	Orthogonal init (all linear + embeddings)	–

Table 7: $\mathbf{RL}_{\text{b}}$ (baseline) network architecture.

(24 GB). The second machine has an Intel Core i9-9900K CPU @ 3.60 GHz (8 cores/16 threads) and 32 GB RAM. All results in Tables 1 and 3 were obtained on the first workstation. Behavior Tree execution is implemented using the `py_trees` (Stonier et al., 2025) library.

A.4.4 Hyperparameters.

The principal hyperparameters used for training and evaluation are summarized in Table 8. These include model parameters, optimization settings, RL-specific coefficients, supervised learning configurations, evaluation, and expert parameters.

Category	Parameter	Value
General	Image Size	128
	Image Num. Channels	3
	Image Feature Dim.	64
	Text Feature Dim.	32
	Actor/Critic Dim.	64
	Frame Seq. Length	1
	Frameskip	1
Training (RL)	Epochs	4
	Batch Size	256
	Iterations	200 (NL, BJ) 500 (Others)
	Max Episode Steps	100
	Rollout Steps	$T = 2000$ (env steps; factorized RL) $K = \sum_{t=1}^{T} n_t$ (b-RL, baseline)
	Learning Rate (LR)	5×10^{-5}
	LR Scheduler	True
b-RL/RL Loss	Discount Factor γ	0.99
	GAE λ	0.95
	Value Loss Coef.	0.5
	Entropy Coef.	0.05
	Clip Epsilon	0.2
	Cross-Entropy Coef.	0.1 or 1.0
Supervised	Sup. Batch Size	64
	Sup. Epochs	100
	Sup. Patience	30
	Sup. Learning Rate	1×10^{-4}
DAgger	Iteration Budget Quota	100 (NL, FL4x4) 1,000 (Others)
	Initial Beta	1.0
	Decay α	0.95
Evaluation	Eval. Episodes	100
	Eval. Interval	10
	Eval. Greedy	True
Expert	Expert Budget (API Calls)	1,000 (NL, FL4x4) 10,000 (Others)

Table 8: Key hyperparameters used in all experiments.

A.5 Experimental Environments

GymCards. We evaluate our method on visual reasoning and decision-making tasks from the GymCards suite (Zhai et al., 2024): *NumberLine*, *EZPoints*, and *BlackJack*.

In *NumberLine*, the agent operates in a deterministic environment and must increment or decrement a visual counter to match a target number, requiring basic numeric comparison. The reward is $+1$ when the agent reaches the target, 0 when the agent gets closer to the target without finishing the task, and -1 when the agent stays equally far from the target or moves farther away. Episodes terminate upon reaching the target value or when a maximum step limit is reached.

EZPoints is a visual arithmetic task in which the agent is shown two playing cards, each displaying a number, and must use arithmetic operators to build an expression that equals a fixed target value (e.g., 12). Each card is used once, demanding both visual perception and symbolic reasoning. Valid intermediate actions receive reward 0. When the agent selects "=", the episode terminates with reward $+10$ if the constructed expression evaluates to the target and uses both cards exactly once, and -1 otherwise. Invalid card selections also receive -1, and exceeding the internal formula-length limit truncates the episode with reward -1. Figure 6 illustrates an example of a step-by-step progression in *EZPoints*.

BlackJack is a stochastic version of the classic card game, where the agent decides whether to "hit" or "stand" based on visually observed cards, with uncertainty introduced by hidden cards and random draws. A non-busting "hit" receives reward 0, busting gives -1, and choosing "stand" ends the

episode with terminal reward $+1$ for beating the dealer, 0 for a draw, and -1 for a loss.

FrozenLake. We assess our approach on two visual navigation tasks based on the classic FrozenLake environment in *4x4* and *8x8* random grid settings (Towers et al., 2024). In each episode, the agent is presented with a top-down grid map and must navigate from a start position to a designated goal, avoiding "holes" that terminate the episode. Rewards are sparse and native to FrozenLake: entering the goal tile yields $+1$, while all other transitions, including moves onto ordinary frozen tiles, falling into a hole, and time-limit truncation, yield 0. Episodes terminate upon reaching the goal or falling into a hole, and are truncated upon reaching a predefined maximum step limit. The environment is fully observable and deterministic, but the larger *8x8* grid increases task complexity by introducing a greater number of possible paths and obstacles. Both variants require spatial reasoning and safe action selection, with the *8x8* setting posing a significantly greater challenge in terms of path diversity and episode length, under the sparse-reward settings described earlier.

BabyAIText. We evaluate our method on the BabyAIText environment (Paglieri et al., 2025), a two-dimensional grid world where observations are provided as rendered RGB images rather than symbolic states, along with natural language instructions describing the task. The agent must follow these instructions to achieve specific goals. We focus on two task variants: *Goto* and *Pickup*. In the *Goto* task, the agent is instructed to navigate to a target object described in natural language (e.g., "go to the blue ball"). In the *Pickup* task, the agent must locate and pick up a specified object (e.g., "pick up the grey key").

In our framework, the compact model operates on the rendered image together with condition-query embeddings and does not directly process the instruction text. Instead, the instruction is implicitly encoded through the behavior tree design and its associated condition queries. Both *Goto* and *Pickup* use the native BabyAI success reward: successful completion yields $1-0.9\cdot(t/T)$, where t is the elapsed step count and T is the environment's maximum step budget, so faster solutions receive higher reward. All non-success transitions, failures, and timeouts receive 0. Episodes terminate upon successful completion of the instruction or when the task fails, and are truncated upon reaching a predefined maximum step limit. Both tasks require grounding language in visual perception, spatial reasoning, and sequential planning. Each episode is generated with random object types and colors to provide a diverse and challenging set of scenarios for both training and evaluation.

A.6 Prompts for Expert Data Acquisition

NumberLine

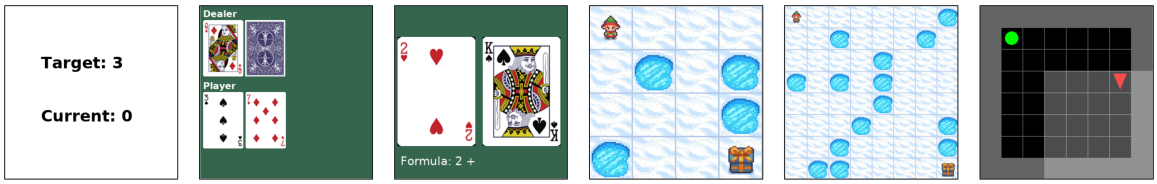
Prompt:

```
Game: NumberLine-v0 - Visual Number Line
↪  Game (goal: reach the target
↪  position).

Observation:
- The screen displays a horizontal number
↪  line.
- The 'target position' and the 'current
↪  position' are indicated by labeled
↪  text rendered along this line.
- There is no direct access to the numeric
↪  positions—you must infer the current
↪  and target locations by reading the
↪  text in the image.

Condition Under Evaluation:
↪  '<node.function>'
Your task is to determine whether this
↪  condition is TRUE or FALSE based on
↪  the current visual observation.

Previous Reasoning:
<previous_thought>

Reasoning Guidelines:
1. Read and interpret the labeled 'Target'
↪  and 'Current' positions shown on the
↪  number line.
2. Determine whether the current position
↪  is to the left or right of the target.
3. Use this understanding to evaluate
↪  whether the condition logically
↪  matches what is visible.
4. Only return 'True' if the image clearly
↪  supports the condition being
↪  evaluated.

Respond ONLY in the following JSON format:
{
  "thought": "<Step-by-step reasoning>",
  "action": "<True or False>"
}
```

Blackjack

Prompt:

```
Game: Blackjack-v0 - Visual Blackjack Game
↪  (goal: get closer to 21 than the
↪  dealer).

Observation:
```



NumberLine | BlackJack | EZPoints | FrozenLake 4x4 | FrozenLake 8x8 | BabyAIText Goto/Pickup

Figure 5: Illustrations of the experimental environments. From left to right: (1) **NumberLine**, increment or decrement the counter to match a target; (2) **BlackJack**, decide to "hit" or "stand" based on visible and hidden cards; (3) **EZPoints**, build an arithmetic expression from shown cards to match a target value; (4) **FrozenLake 4x4**, navigate a small frozen grid to the goal while avoiding holes; (5) **FrozenLake 8x8**, a larger and more challenging variant; (6) **BabyAIText Goto/Pickup**, perform language-guided navigation and object manipulation in a grid world.

(a) Initial step: No tokens selected. (b) After selecting digit '2'. (c) After selecting '+'.

Figure 6: Example trajectory in *EZPoints*. The agent constructs an arithmetic formula step-by-step using visual input. It first selects a digit (b), then an operator (c), and continues until the expression reaches the target value (e.g., 12). The final step involves selecting the equals token (=) to submit the completed formula.

```
- The top part of the image shows the
↪  dealer's visible card.
- The bottom part shows the player's
↪  current hand.
- There is no numerical input. You must
↪  infer hand values from card visuals.
- Face cards (J, Q, K) count as 10. Ace
↪  can count as either 1 or 11.

Condition Under Evaluation:
↪  '<node.function>'
Your task is to determine whether this
↪  condition is TRUE or FALSE in the
↪  current state.

Previous Reasoning:
<previous_thought>

Reasoning Guidelines:
1. Identify the dealer's visible card and
↪  assess its strength.
2. Visually estimate the player's hand
↪  value using the visible cards.
3. Evaluate whether the condition
↪  description logically matches the
↪  current visual state.
4. Only return 'True' if the visual
↪  evidence clearly supports the
↪  condition.

Respond ONLY in the following JSON format:
```

```
{
  "thought": "<Step-by-step reasoning>",
  "action": "<True or False>"
}
```

EZPoints

Prompt:

```
Game: EZPoints-v0 - Card-Based Arithmetic
↪  Game (goal: evaluate to exactly 12).

Objective:
- Construct a valid arithmetic expression
↪  by selecting cards one at a time.
- Expressions must use visible cards
↪  (digits 1-10) and operators (+, *, =).
- Use '=' only if the formula is complete
↪  and expected to evaluate to 12.
- A maximum of 5 tokens can be selected
↪  (e.g., '4 + 8').

Game Rules:
- Number cards must match one of the
↪  visible cards and cannot repeat.
- Operators can only appear between valid
↪  numbers and must follow the formula
↪  logic.
```

```
- '=' is only valid if the formula
↪  includes at least two numbers and one
↪  operator.
- Face cards (J, Q, K) count as 10.

Rewards:
- +10 if '=' is selected and formula
↪  evaluates to exactly 12.
- −1 for invalid actions (e.g., repeated
↪  card, premature '=')
-  0 for valid intermediate steps.

Condition Under Evaluation:
↪  '<node.function>'
Your task is to determine whether this
↪  condition is TRUE or FALSE in the
↪  current state.

Previous Reasoning:
<previous_thought>

Reasoning Guidelines:
1. Assess if the condition logically fits
↪  the current formula and visible cards.
2. Only return 'True' if the condition
↪  leads to a valid and useful next step.
3. Use visual and textual cues from the
↪  image (card layout, formula progress).

Respond ONLY in the following JSON format:
{
  "thought": "<Step-by-step reasoning>",
  "action": "<True or False>"
}
```

FrozenLake 4x4

Prompt:

```
Game: FrozenLake-v1 on a 4x4 grid (goal at
↪  (3,3)).
Coordinate System: (0, 0) is top-left.
↪  Format is (row, column).
Row increases ↓, Column increases →.

Grid Information:
- S = Start (safe),
- F = Frozen tile (safe),
- H = Hole (dangerous),
- G = Goal (target destination).

- (0,0)=S, (0,1)=F, (0,2)=F, (0,3)=H
- (1,0)=F, (1,1)=F, (1,2)=F, (1,3)=H
- (2,0)=F, (2,1)=H, (2,2)=F, (2,3)=H
- (3,0)=H, (3,1)=F, (3,2)=F, (3,3)=G
- Agent is currently at (row <row>, column
↪  <col>), standing on tile '<tile>'.

Objective:
- Reach the goal tile (G) at (3,3) safely
↪  and efficiently.

Previous Actions:
- <previous 4 actions>

Reasoning Guidelines:
- Only RIGHT, DOWN, LEFT, or UP moves are
↪  allowed at each step (diagonal or
↪  jumping moves are not permitted).
Question: Based on the above, is the
↪  condition '<node.function>' true?

Respond ONLY in the following JSON format:
{
  "thought": "<Step-by-step justification
  ↪  based on reasoning guidelines>",
  "action": "<True or False>"
}
```

FrozenLake 8x8

Prompt:

```
Game: FrozenLake-v1 on an 8×8 grid (goal
↪  at (7,7)).
Coordinate System: (0, 0) is top-left.
↪  Format is (row, column).
Row increases ↓, Column increases →.

Grid Information:
- S = Start (safe),
- F = Frozen tile (safe),
- H = Hole (dangerous),
- G = Goal (target destination).

- (0,0)=S, (0,1)=F, (0,2)=F, (0,3)=F,
↪  (0,4)=F, (0,5)=F, (0,6)=F, (0,7)=H
- (1,0)=F, (1,1)=F, (1,2)=H, (1,3)=F,
↪  (1,4)=F, (1,5)=H, (1,6)=F, (1,7)=H
- (2,0)=F, (2,1)=F, (2,2)=F, (2,3)=F,
↪  (2,4)=H, (2,5)=F, (2,6)=F, (2,7)=F
- (3,0)=H, (3,1)=H, (3,2)=F, (3,3)=H,
↪  (3,4)=F, (3,5)=F, (3,6)=F, (3,7)=H
- (4,0)=F, (4,1)=F, (4,2)=F, (4,3)=F,
↪  (4,4)=F, (4,5)=H, (4,6)=F, (4,7)=F
- (5,0)=F, (5,1)=H, (5,2)=F, (5,3)=F,
↪  (5,4)=F, (5,5)=F, (5,6)=H, (5,7)=F
- (6,0)=F, (6,1)=F, (6,2)=F, (6,3)=F,
↪  (6,4)=H, (6,5)=F, (6,6)=F, (6,7)=H
- (7,0)=F, (7,1)=H, (7,2)=F, (7,3)=H,
↪  (7,4)=F, (7,5)=F, (7,6)=F, (7,7)=G
- Agent is currently at (row <row>, column
↪  <col>), standing on tile '<tile>'.

Objective:
- Reach the goal tile (G) at (7,7) safely
↪  and efficiently.

Previous Actions:
- <previous 4 actions>

Reasoning Guidelines:
- Only RIGHT, DOWN, LEFT, or UP moves are
↪  allowed at each step (diagonal or
↪  jumping moves are not permitted).
Question: Based on the above, is the
↪  condition '<node.function>' true?

Respond ONLY in the following JSON format:
{
  "thought": "<Step-by-step justification
  ↪  based on reasoning guidelines>",
  "action": "<True or False>"
```

```
}
```

BabyAIText Goto

Prompt:

```
Game: BabyAIText-v0 - Grid World with
↪  Language Instructions.

Mission: <mission text>
Coordinate System: (0, 0) is top-left.
↪  Format is (row, column).
Row increases ↓, Column increases →.
Agent Position: <row, col>, Facing:
↪  <direction>
Visible Objects:
- <object list, e.g., "green box at (1,2)"
↪  ...>

Scene Description: <long term context from
↪  environment>
Possible Actions: <action_1>, <action_2>,
↪  ...

Condition Under Evaluation:
↪  '<node.function>'
Your task is to determine whether this
↪  condition is TRUE or FALSE in the
↪  current state.

Reasoning Guidelines:
0. The agent is shaped like a triangle.
↪  The pointed tip of the triangle
↪  indicates the direction it is facing
↪  (e.g., North, South, East, West).
↪  'Turn left' means rotating the
↪  triangle (agent) 90 degrees
↪  counterclockwise, and 'Turn right'
↪  means rotating it 90 degrees
↪  clockwise, relative to the direction
↪  the tip is pointing.
1. Directional terms like 'Left' and
↪  'Right' are defined strictly relative
↪  to the agent's current facing
↪  direction. This means:
   - If facing North, 'left' is West and
   ↪  'right' is East.
   - If facing East, 'left' is North and
   ↪  'right' is South.
   - If facing South, 'left' is East and
   ↪  'right' is West.
   - If facing West, 'left' is South and
   ↪  'right' is North.
   * Do not confuse these with object
   ↪  positions on the map or coordinates
   ↪  like (x, y). Always interpret
   ↪  directions from the agent's
   ↪  perspective.
   For example, if the agent is facing
   ↪  South and an object is at (x+1, y),
   ↪  that object is to the agent's left.
2. Explore the area if there are no target
↪  objects mentioned in the Scene
↪  Description.

Respond ONLY in the following JSON format:
{
  "thought": "<Step-by-step reasoning>",
  "action": "<True or False>"
}
```

BabyAIText Pickup

Prompt:

```
Game: BabyAIText-v0 - Grid World with
↪  Language Instructions.

Mission: <mission text>
Coordinate System: (0, 0) is top-left.
↪  Format is (row, column).
Row increases ↓, Column increases →.
Agent Position: <row, col>, Facing:
↪  <direction>
Visible Objects:
- <object list, e.g., "green box at (1,2)"
↪  ...>

Scene Description: <long term context from
↪  environment>
Possible Actions: <action_1>, <action_2>,
↪  ...

Condition Under Evaluation:
↪  '<node.function>'
Your task is to determine whether this
↪  condition is TRUE or FALSE in the
↪  current state.

Reasoning Guidelines:
0. The agent is shaped like a triangle.
↪  The pointed tip of the triangle
↪  indicates the direction it is facing
↪  (e.g., North, South, East, West).
↪  'Turn left' means rotating the
↪  triangle (agent) 90 degrees
↪  counterclockwise, and 'Turn right'
↪  means rotating it 90 degrees
↪  clockwise, relative to the direction
↪  the tip is pointing.
1. An object can only be picked up if it
↪  is in the cell directly in front of
↪  the agent.
2. Some objects, such as doors, cannot be
↪  picked up. Only items like keys,
↪  boxes, or balls are pickable.
3. Directional terms like 'Left' and
↪  'Right' are defined strictly relative
↪  to the agent's current facing
↪  direction. This means:
   - If facing North, 'left' is West and
   ↪  'right' is East.
   - If facing East, 'left' is North and
   ↪  'right' is South.
   - If facing South, 'left' is East and
   ↪  'right' is West.
   - If facing West, 'left' is South and
   ↪  'right' is North.
   * Do not confuse these with object
   ↪  positions on the map or coordinates
   ↪  like (x, y). Always interpret
   ↪  directions from the agent's
   ↪  perspective.
```

```
4. Explore the area if there are no target objects mentioned in the Scene Description.

Respond ONLY in the following JSON format:
{
  "thought": "<Step-by-step reasoning>",
  "action": "<True or False>"
}
```

A.7 Bahavior Trees for Each Task

NumberLine

Behavior Tree:

```
{
    "type": "selector",
    "name": "Root",
    "children": [
        {
            "type": "sequence",
            "name": "Move Right Sequence",
            "children": [
                {
                    "type": "condition",
                    "name": "Check Current Less Than Target",
                    "function": "Is current position less than target?"
                },
                {
                    "type": "action",
                    "name": "Execute Move Right",
                    "function": "MOVE_RIGHT"
                }
            ]
        },
        {
            "type": "sequence",
            "name": "Move Left Sequence",
            "children": [
                {
                    "type": "condition",
                    "name": "Check Current Greater Than Target",
                    "function": "Is current position greater than target?"
                },
                {
                    "type": "action",
                    "name": "Execute Move Left",
                    "function": "MOVE_LEFT"
                }
            ]
        },
        {
            "type": "action",
            "name": "Default Move Right",
            "function": "MOVE_RIGHT"
        }
    ]
}
```

Blackjack

Behavior Tree:

```
{
  "type": "selector",
  "name": "Root",
  "children": [
    {
      "type": "sequence",
      "name": "Stick If Hand Is Exactly 21",
      "children": [
        {
          "type": "condition",
          "name": "Hand Equals 21",
          "function": "Is the player's hand exactly 21?"
        },
        {
          "type": "action",
          "name": "Select Stick",
          "function": "STICK"
        }
      ]
    },
    {
      "type": "sequence",
      "name": "Stick If Hand Is Very High",
      "children": [
        {
          "type": "condition",
          "name": "Hand Is 19 or 20",
          "function": "Is the player's hand value very high (19 or 20)?"
        },
        {
          "type": "action",
          "name": "Select Stick",
          "function": "STICK"
        }
      ]
    },
    {
      "type": "sequence",
      "name": "Hit If Dealer Strong And Hand Moderate",
      "children": [
        {
          "type": "condition",
          "name": "Dealer Has Strong Card",
          "function": "Is the dealer's visible card strong (7 to Ace)?"
        },
        {
          "type": "condition",
          "name": "Hand Is 17 or 18",
          "function": "Is the player's hand in moderate range (17 or 18)?"
        },
        {
          "type": "action",
          "name": "Select Hit",
```

```
          "function": "HIT"
        }
      ]
    },
    {
      "type": "sequence",
      "name": "Hit If Hand Low",
      "children": [
        {
          "type": "condition",
          "name": "Hand Below 17",
          "function": "Is the player's
          ↪  hand below 17?"
        },
        {
          "type": "action",
          "name": "Select Hit",
          "function": "HIT"
        }
      ]
    },
    {
      "type": "action",
      "name": "Fallback Stick",
      "function": "STICK"
    }
  ]
}
```

EZPoints

Behavior Tree:

```
{
"type": "selector",
"name": "Root",
"children": [
    {
    "type": "sequence",
    "name": "Try Finish Formula
    ↪  Correctly",
    "children": [
        {
        "type": "condition",
        "name": "Formula Ready Equals 12",
        "function": "Is the formula
        ↪  complete and does it evaluate
        ↪  to 12?"
        },
        {
        "type": "action",
        "name": "Select Equals",
        "function": "="
        }
    ]
    },
    {
    "type": "sequence",
    "name": "Add Operator If Needed",
    "children": [
        {
        "type": "condition",
        "name": "Needs Operator",
        "function": "Does the formula end
        ↪  with a number and not yet have
        ↪  an operator?"
        },
        {
        "type": "selector",
        "name": "Choose Operator",
        "children": [
            {
            "type": "sequence",
            "name": "Try Plus",
            "children": [
                {
                "type": "condition",
                "name": "Should Use Plus",
                "function": "Is '+' a
                ↪  better choice based
                ↪  on remaining values?"
                },
                {
                "type": "action",
                "name": "Select Plus",
                "function": "+"
                }
            ]
            },
            {
            "type": "action",
            "name": "Select Multiply",
            "function": "*"
            }
        ]
        }
    ]
    },
    {
    "type": "sequence",
    "name": "Add Number If Needed",
    "children": [
        {
        "type": "condition",
        "name": "Needs Number",
        "function": "Is the formula empty
        ↪  or ends with an operator?"
        },
        {
        "type": "selector",
        "name": "Choose Number From
        ↪  Visible Cards",
        "children": [
            {
            "type": "sequence",
            "name": "Use 1",
            "children": [
                {
                "type": "condition",
                "name": "Can Use 1",
                "function": "Is '1'
                ↪  visible in cards and
                ↪  not in the formula?"
                },
                {
                "type": "action",
                "name": "Select 1",
                "function": "1"
                }
            ]
            },
            {
            "type": "sequence",
            "name": "Use 2",
            "children": [
```

```
            {
            "type": "condition",
            "name": "Can Use 2",
            "function": "Is '2'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 2",
            "function": "2"
            }
        ]
        },
        {
        "type": "sequence",
        "name": "Use 3",
        "children": [
            {
            "type": "condition",
            "name": "Can Use 3",
            "function": "Is '3'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 3",
            "function": "3"
            }
        ]
        },
        {
        "type": "sequence",
        "name": "Use 4",
        "children": [
            {
            "type": "condition",
            "name": "Can Use 4",
            "function": "Is '4'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 4",
            "function": "4"
            }
        ]
        },
        {
        "type": "sequence",
        "name": "Use 5",
        "children": [
            {
            "type": "condition",
            "name": "Can Use 5",
            "function": "Is '5'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 5",
            "function": "5"
            }
        ]
        },
```

```
        {
        "type": "sequence",
        "name": "Use 6",
        "children": [
            {
            "type": "condition",
            "name": "Can Use 6",
            "function": "Is '6'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 6",
            "function": "6"
            }
        ]
        },
        {
        "type": "sequence",
        "name": "Use 7",
        "children": [
            {
            "type": "condition",
            "name": "Can Use 7",
            "function": "Is '7'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 7",
            "function": "7"
            }
        ]
        },
        {
        "type": "sequence",
        "name": "Use 8",
        "children": [
            {
            "type": "condition",
            "name": "Can Use 8",
            "function": "Is '8'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 8",
            "function": "8"
            }
        ]
        },
        {
        "type": "sequence",
        "name": "Use 9",
        "children": [
            {
            "type": "condition",
            "name": "Can Use 9",
            "function": "Is '9'
            ↪  visible in cards and
            ↪  not in the formula?"
            },
            {
            "type": "action",
            "name": "Select 9",
```

```
                "function": "9"
                }
            ]
            },
            {
            "type": "sequence",
            "name": "Use 10",
            "children": [
                {
                "type": "condition",
                "name": "Can Use 10",
                "function": "Is '10'
                ↪  visible in cards and
                ↪  not in the formula?"
                },
                {
                "type": "action",
                "name": "Select 10",
                "function": "10"
                }
            ]
            }
        ]
        }
    ]
    },
    {
    "type": "sequence",
    "name": "Force Finish If Out Of
    ↪  Tokens",
    "children": [
        {
        "type": "condition",
        "name": "Formula At Max Tokens",
        "function": "Is the formula at 5
        ↪  tokens and not yet evaluated?"
        },
        {
        "type": "action",
        "name": "Force Equals",
        "function": "="
        }
    ]
    },
    {
    "type": "action",
    "name": "Default Select Equals",
    "function": "="
    }
]
}
```

FrozenLake 4x4 / 8x8

Behavior Tree:

```
{
  "type": "selector",
  "name": "Root",
  "children": [
    {
      "type": "sequence",
      "name": "Move RIGHT if Safe and
      ↪  Valid Path",
      "children": [
        {
          "type": "condition",
          "name": "RIGHT is non-hole",
          "function": "Does moving RIGHT
          ↪  lead to a non-hole tile?"
        },
        {
          "type": "condition",
         "name": "RIGHT is on valid path",
          "function": "Is the next tile
          ↪  after moving RIGHT not
          ↪  surrounded on three sides by
          ↪  holes or by holes and walls,
          ↪  can you immediately continue
          ↪  moving down or right from
          ↪  there to reach the goal, and
          ↪  does this move avoid
          ↪  repeatedly alternating
          ↪  between two actions (such
          ↪  as [RIGHT, LEFT] followed by
          ↪  another RIGHT)?"
        },
        {
          "type": "action",
          "name": "Move RIGHT",
          "function": "RIGHT"
        }
      ]
    },
    {
      "type": "sequence",
      "name": "Move DOWN if Safe and Valid
      ↪  Path",
      "children": [
        {
          "type": "condition",
          "name": "DOWN is non-hole",
          "function": "Does moving DOWN
          ↪  lead to a non-hole tile?"
        },
        {
          "type": "condition",
         "name": "DOWN is on valid path",
          "function": "Is the next tile
          ↪  after moving DOWN not
          ↪  surrounded on three sides by
          ↪  holes or by holes and walls,
          ↪  can you immediately
          ↪  continue moving down or
          ↪  right from there to reach
          ↪  the goal, and does this move
          ↪  avoid repeatedly
          ↪  alternating between two
          ↪  actions (such as [DOWN, UP]
          ↪  followed by another DOWN)?"
        },
        {
          "type": "action",
          "name": "Move DOWN",
          "function": "DOWN"
        }
      ]
    },
    {
      "type": "sequence",
      "name": "Move LEFT if Safe and Valid
      ↪  Path",
      "children": [
        {
          "type": "condition",
```

```
          "name": "LEFT is non-hole",
          "function": "Does moving LEFT
          ↪  lead to a non-hole tile?"
        },
        {
          "type": "condition",
          "name": "LEFT is on valid path",
          "function": "Is the next tile
          ↪  after moving LEFT not
          ↪  surrounded on three sides by
          ↪  holes or by holes and walls,
          ↪  can you immediately continue
          ↪  moving down or right from
          ↪  there to reach the goal, and
          ↪  does this move avoid
          ↪  repeatedly alternating
          ↪  between two actions (such
          ↪  as [LEFT, RIGHT] followed by
          ↪  another LEFT)?"
        },
        {
          "type": "action",
          "name": "Move LEFT",
          "function": "LEFT"
        }
      ]
    },
    {
      "type": "sequence",
      "name": "Move UP if Safe and Valid
      ↪  Path",
      "children": [
        {
          "type": "condition",
          "name": "UP is non-hole",
          "function": "Does moving UP lead
          ↪  to a non-hole tile?"
        },
        {
          "type": "condition",
          "name": "UP is on valid path",
          "function": "Is the next tile
          ↪  after moving UP not
          ↪  surrounded on three sides by
          ↪  holes or by holes and walls,
          ↪  can you immediately
          ↪  continue moving down or
          ↪  right from there to reach
          ↪  the goal, and does this move
          ↪  avoid repeatedly
          ↪  alternating between two
          ↪  actions (such as [UP, DOWN]
          ↪  followed by another UP)?"
        },
        {
          "type": "action",
          "name": "Move UP",
          "function": "UP"
        }
      ]
    },
    {
      "type": "action",
      "name": "Default Action (LEFT)",
      "function": "LEFT"
    }
  ]
}
```

BabyAIText Goto

Behavior Tree:

```
{
  "type": "selector",
  "name": "Root",
  "children": [
    {
      "type": "sequence",
      "name": "Move Forward Toward
      ↪  Target",
      "children": [
        {
          "type": "condition",
          "name": "Target Ahead",
          "function": "Is the target
          ↪  object located somewhere in
          ↪  front of the agent (not 0
          ↪  steps forward), possibly
          ↪  with a few steps left or
          ↪  right relative to its
          ↪  facing direction? (e.g., x
          ↪  steps left/right and y
          ↪  steps forward)?"
        },
        {
          "type": "condition",
          "name": "Immediate Step Clear",
          "function": "Is the cell one
          ↪  step ahead of the agent
          ↪  clear (i.e., no object
          ↪  listed at that position in
          ↪  'Visible Objects')?"
        },
        {
          "type": "action",
          "name": "Move Forward",
          "function": "go forward"
        }
      ]
    },

    {
      "type": "sequence",
      "name": "Rotate Left Toward Target",
      "children": [
        {
          "type": "condition",
          "name": "Need To Turn Left",
          "function": "Is the target object
          ↪  located to the agent's left
          ↪  side (0~90 degrees
          ↪  counterclockwise) from its
          ↪  current facing direction?"
        },
        {
          "type": "action",
          "name": "Turn Left",
          "function": "turn left"
        }
      ]
    },

    {
      "type": "sequence",
      "name": "Rotate Right Toward
      ↪  Target",
      "children": [
```

```
        {
          "type": "condition",
          "name": "Need To Turn Right",
          "function": "Is the target
          ↪  object located to the
          ↪  agent's right side (0~90
          ↪  degrees clockwise) from its
          ↪  current facing direction?"
        },
        {
          "type": "action",
          "name": "Turn Right",
          "function": "turn right"
        }
      ]
    },

    {
      "type": "selector",
      "name": "Exploration (Optimal
      ↪  Default Action)",
      "children": [
        {
          "type": "sequence",
          "name": "Explore Left Required",
          "children": [
            {
              "type": "condition",
              "name": "Turn Left Toward
              ↪  Target?",
              "function": "Is turning left
              ↪  (90 degrees
              ↪  counterclockwise,
              ↪  possibly multiple times)
              ↪  required among the
              ↪  possible actions to
              ↪  reach the target
              ↪  object?"
            },
            {
              "type": "action",
              "name": "Turn Left",
              "function": "turn left"
            }
          ]
        },
        {
          "type": "sequence",
          "name": "Explore Right Required",
          "children": [
            {
              "type": "condition",
              "name": "Turn Right Toward
              ↪  Target?",
              "function": "Is turning right
              ↪  (90 degrees clockwise,
              ↪  possibly multiple times)
              ↪  required among the
              ↪  possible actions to
              ↪  reach the target
              ↪  object?"
            },
            {
              "type": "action",
              "name": "Turn Right",
              "function": "turn right"
            }
          ]
        },
        {
          "type": "sequence",
          "name": "Explore Forward
          ↪  Required",
          "children": [
            {
              "type": "condition",
              "name": "Go Forward Toward
              ↪  Target?",
              "function": "Is going
              ↪  forward required among
              ↪  the possible actions to
              ↪  reach the target
              ↪  object?"
            },
            {
              "type": "action",
              "name": "Go Forward",
              "function": "go forward"
            }
          ]
        }
      ]
    },
    {
      "type": "action",
      "name": "Default Action (Turn
      ↪  Left)",
      "function": "turn left"
    }
  ]
}
```

BabyAIText Pickup

Behavior Tree:

```
{
  "type": "selector",
  "name": "Root",
  "children": [
    {
      "type": "sequence",
      "name": "Pick Up Target If In Front",
      "children": [
        {
          "type": "condition",
          "name": "Target Directly Ahead",
          "function": "Is the target object
          ↪  located in the cell directly
          ↪  in front of the agent?"
        },
        {
          "type": "action",
          "name": "Pick Up",
          "function": "pick up"
        }
      ]
    },

    {
      "type": "sequence",
      "name": "Move Forward Toward
      ↪  Target",
      "children": [
        {
```

```
          "type": "condition",
          "name": "Target Ahead",
          "function": "Is the target
          ↪  object located somewhere in
          ↪  front of the agent (not 0
          ↪  steps forward), possibly
          ↪  with a few steps left or
          ↪  right relative to its
          ↪  facing direction? (e.g., x
          ↪  steps left/right and y
          ↪  steps forward)?"
        },
        {
          "type": "condition",
          "name": "Immediate Step Clear",
          "function": "Is the cell one
          ↪  step ahead of the agent
          ↪  clear (i.e., no object
          ↪  listed at that position in
          ↪  'Visible Objects')?"
        },
        {
          "type": "action",
          "name": "Move Forward",
          "function": "go forward"
        }
      ]
    },

    {
      "type": "sequence",
      "name": "Rotate Left Toward Target",
      "children": [
        {
          "type": "condition",
          "name": "Need To Turn Left",
          "function": "Is the target object
          ↪  located to the agent's left
          ↪  side (0~90 degrees
          ↪  counterclockwise) from its
          ↪  current facing direction?"
        },
        {
          "type": "action",
          "name": "Turn Left",
          "function": "turn left"
        }
      ]
    },

    {
      "type": "sequence",
      "name": "Rotate Right Toward
      ↪  Target",
      "children": [
        {
          "type": "condition",
          "name": "Need To Turn Right",
          "function": "Is the target
          ↪  object located to the
          ↪  agent's right side (0~90
          ↪  degrees clockwise) from its
          ↪  current facing direction?"
        },
        {
          "type": "action",
          "name": "Turn Right",
          "function": "turn right"
        }
      ]
    },

    {
      "type": "selector",
      "name": "Exploration (Optimal
      ↪  Default Action)",
      "children": [
        {
          "type": "sequence",
          "name": "Explore Left Required",
          "children": [
            {
              "type": "condition",
              "name": "Turn Left Toward
              ↪  Target?",
              "function": "Is turning left
              ↪  (90 degrees
              ↪  counterclockwise,
              ↪  possibly multiple times)
              ↪  required among the
              ↪  possible actions to
              ↪  reach the target
              ↪  object?"
            },
            {
              "type": "action",
              "name": "Turn Left",
              "function": "turn left"
            }
          ]
        },
        {
          "type": "sequence",
          "name": "Explore Right Required",
          "children": [
            {
              "type": "condition",
              "name": "Turn Right Toward
              ↪  Target?",
              "function": "Is turning right
              ↪  (90 degrees clockwise,
              ↪  possibly multiple times)
              ↪  required among the
              ↪  possible actions to
              ↪  reach the target
              ↪  object?"
            },
            {
              "type": "action",
              "name": "Turn Right",
              "function": "turn right"
            }
          ]
        },
        {
          "type": "sequence",
          "name": "Explore Forward
          ↪  Required",
          "children": [
            {
              "type": "condition",
              "name": "Go Forward Toward
              ↪  Target?",
```

```
              "function": "Is going
              ↪  forward required among
              ↪  the possible actions to
              ↪  reach the target
              ↪  object?"
            },
            {
              "type": "action",
              "name": "Go Forward",
              "function": "go forward"
            }
          ]
        }
      ]
    },
    {
      "type": "action",
      "name": "Default Action (Turn
      ↪  Left)",
      "function": "turn left"
    }
  ]
}
```


A.8 Visualization of Behavior Trees for each task

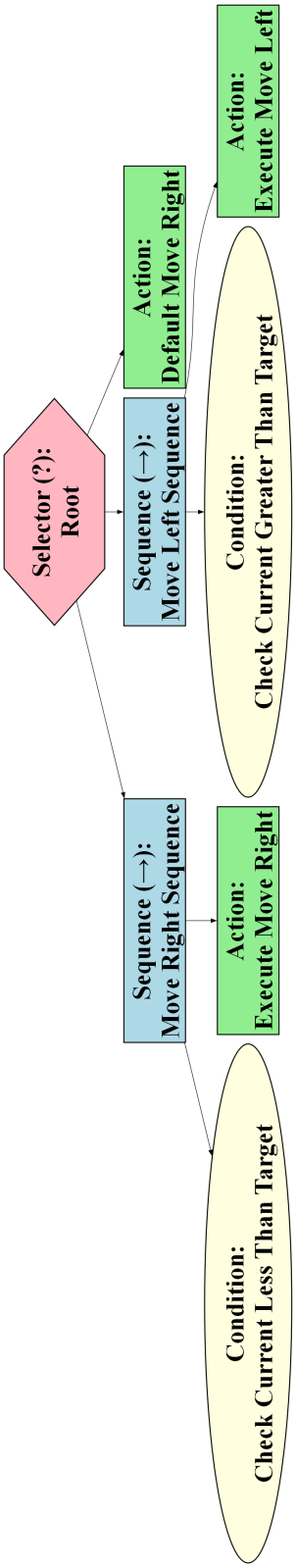


Figure 7: NumberLine

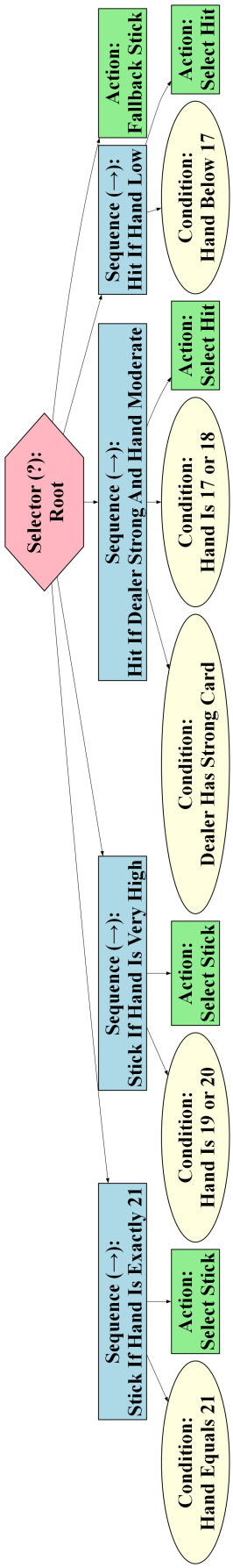


Figure 8: BlackJack

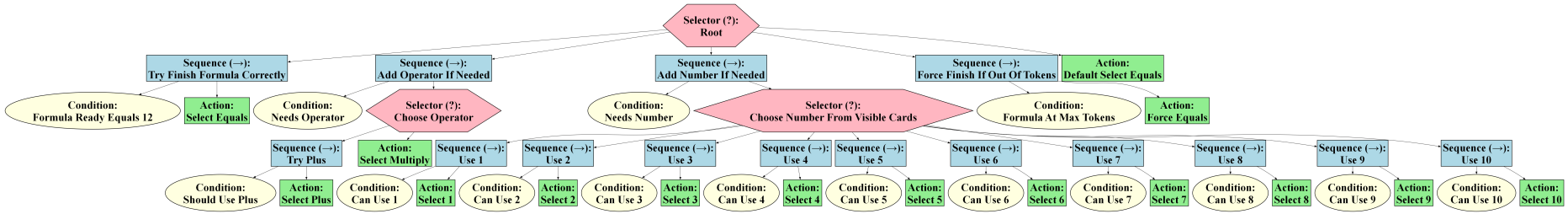


Figure 41: EZ Points

Figure 10: FrozenLake 4x4 / 8x8

Figure 11: BabyAIText Goto

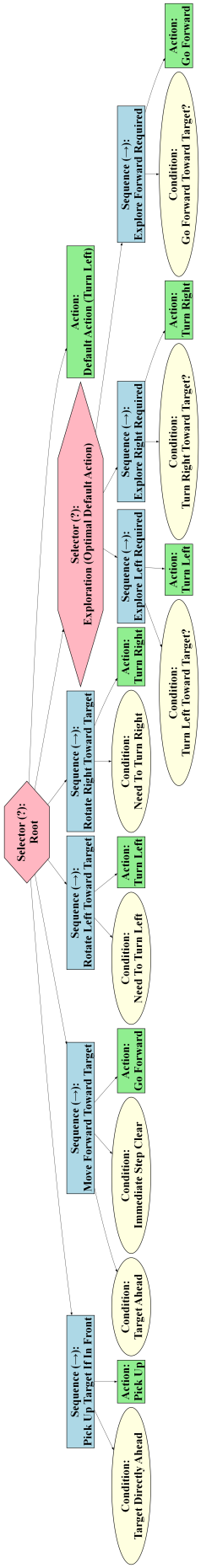


Figure 12: BabyAIText Pickup