Dual-Resolution Recursive Energy: Certified Contract–Expand Inference for Sequential Decision Making

Haozhou Gao [1] **Jiarui Zeng** [2] **Wendi Ren** [1] **Yanwen Liu** [1] **Shuang Li** [1]

Abstract

Fast habits and slow deliberation are usually modeled as separate systems. We propose a different view: **System 1 and System 2 are two resolutions of the same recursive energy**. A candidate decision is represented by a tree-structured energy whose root score determines the preferred action. The tree can be evaluated coarsely, by replacing subtrees with learned macro-energies, or finely, by expanding selected subtrees into their lower-level reasoning steps. In this view, a habit is a certified compressed evaluation, while deliberation is selective expansion of the same decision structure. The model switches resolution through learned certificates rather than a separate heuristic gate. If the contracted energy is sufficient to certify the current action choice, the model acts immediately; otherwise, it expands only the uncertain parts of the tree. Training follows a wake–sleep procedure: wake fits expert actions and records useful expansions, while sleep distills these expanded evaluations into macro-energies and calibrates contraction certificates. At test time, the same certificates guide online contract–expand decision making under a compute budget. This yields an imitation-based model of adaptive deliberation in which reasoning effort is allocated only when needed, while preserving explicit traces of the computation behind each action.

1. Introduction

A central challenge in modeling intelligent behavior is not only predicting what action will be taken, but also capturing how much computation is required to produce it. The same action can require very different amounts of computation depending on context. For example, choosing a familiar route home may be an immediate habitual response, while selecting a detour in an unfamiliar city may require recalling landmarks, evaluating alternatives, and reasoning step by step. A similar distinction appears in sequential control: moving forward in an open corridor and moving forward near a locked door may share the same action label, but require very different internal checks. Models that collapse these cases into a single prediction miss a core property of intelligence: behavior is coupled not only to outcomes, but also to the computation used to produce them.

Prior work has studied this gap from several perspectives, but without a unified computational principle. Dual-process theories distinguish fast and automatic from slow and deliberative cognition, but remain largely descriptive and do not specify how both arise from a single computational mechanism (Sloman, 1996; Evans, 2003; Stanovich & West, 2000; Kahneman, 2011; Evans & Stanovich, 2013). Computational dual-process models introduce explicit interaction between systems, yet still represent them as separate components with an external coupling mechanism rather than a single underlying computation (Sun et al., 2005). In reinforcement learning and control, a related separation appears through cached and goal-directed systems combined via arbitration, which again treats fast and slow behavior as distinct controllers rather than different views of the same decision process (Daw et al., 2005; Dolan & Dayan, 2013; Lee et al., 2014).

In contrast, imitation learning and modern sequence models learn direct mappings from states to actions, but do not represent variation in computational effort, effectively collapsing all decisions into a single fixed-cost prediction (Argall et al., 2009; Ross et al., 2011; Chen et al., 2021; Janner et al., 2021; Shafiullah et al., 2022). Structured and neuro-symbolic methods introduce intermediate reasoning, but typically fix its depth or control it externally, making reasoning an added procedure rather than an emergent property of inference (Manhaeve et al., 2018; Yao et al., 2023). Adaptive computation methods vary the number of steps per input, but treat computation allocation as an external mechanism while leaving the decision representation unchanged (Graves, 2016; Teerapittayanon et al., 2016; Ban-

[1] School of Data Science, The Chinese University of Hong Kong, Shenzhen, China [2] Institute of Collaborative Innovation, University of Macau, Macau SAR, China . Correspondence to: Shuang Li <lishuang@cuhk.edu.cn>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. *Copyright 2026 by the author(s).*

ino et al., 2021). Energy-based models provide a unified scoring framework, but most formulations assign energy only to final outputs or latent variables, leaving the internal structure of computation implicit (LeCun et al., 2006; Belanger & McCallum, 2016; Belanger et al., 2017; Boney et al., 2020; Liu et al., 2021). A more detailed discussion of related work is provided in Appendix A.

Across these perspectives, a common limitation remains: fast and slow computation are either explicitly separated into different mechanisms or controlled externally over a fixed representation. What is missing is a formulation in which computation is not an additional process, but an intrinsic property of the decision representation itself.

This paper takes a different view. We argue that the distinction between System 1 and System 2 is not a distinction between two separate controllers, but between two *resolutions of evaluation*. We use these terms as a computational abstraction: a single model can exhibit both fast and slow behavior depending on how finely it evaluates the same underlying decision structure.

To formalize this idea, we represent each candidate action by an action-conditioned *proof-program tree*, which encodes a latent support structure for why that action should be taken. Internal nodes represent compositional reasoning operations, such as combining requirements or selecting among alternative justifications, while leaves correspond to grounded predicates, subgoal conditions, local action checks, and primitive action consequences. Instead of scoring only the final action, we define a *recursive energy* over the entire tree: subtree energies are composed bottom-up, and the root energy gives the score of the candidate action. A decision is favorable only if the structure that supports it is also low-energy. We argue that *fast and slow behavior emerge from evaluating this same energy at different resolutions, rather than from separate systems or controllers*.

This representation induces two modes of evaluation. Recurrent subtrees can be compressed into macro-energies that summarize their fine-grained structure, enabling fast evaluation and corresponding to habitual behavior. When these coarse evaluations are insufficient, the model expands the relevant subtrees and evaluates them at finer resolution, recovering explicit reasoning. Importantly, both behaviors arise from the *same recursive energy*, differing only in the *resolution* at which the tree is evaluated.

This leads to a principled mechanism for deciding when to deliberate. Each contracted subtree is associated with a learned certificate that estimates its contraction error, i.e., the discrepancy between its coarse macro-energy and its expanded recursive energy. If the coarse evaluation is sufficient to certify the current action choice under the propagated error bounds, computation stops. Otherwise, only uncertain subtrees are expanded. The transition *between fast and slow reasoning is therefore determined internally by the reliability of the model's own evaluations*.

Building on this, we develop a wake–sleep learning framework for imitation-based sequential decision making. Training follows a wake–sleep process on observed expert actions (Hinton et al., 1995). During wake, the model evaluates expert decision points with the current contract–expand mechanism, fits the demonstrated actions, and records which subtrees were expanded. During sleep, these expanded evaluations are distilled into reusable macro-energies, and certificates are calibrated to estimate contraction error.

After training, the learned certificates guide online lazy contract–expand decision making. Candidate actions are first scored in contracted form, and expansion is triggered only when the current best action cannot be certified under the learned error bounds. Thus, online computation is allocated only when the coarse evaluation is not reliable enough for action selection.

Overall, this framework provides a unified view of behavior in which habits correspond to compressed reasoning, deliberation corresponds to selective refinement, and their tradeoff emerges from energy-based certification rather than external control. Unlike prior approaches that separate fast and slow processes, our model explains both within a single representation.

Contributions. This paper makes four contributions: *i*) We formulate fast and slow decision making as two resolutions of a single recursive energy over action-conditioned proof-program trees, rather than as separate policies or controllers. *ii*) We introduce a wake–sleep training objective that combines expert action imitation, macro-energy distillation, and certificate calibration. *iii*) We use the learned certificates for online lazy contract–expand decision making, expanding only uncertified subtrees that could change the current action choice. *iv*) We evaluate the resulting controller on BabyAI and MiniHack, measuring task performance, compositional reuse of grounded decision factors, adaptive computation, and interpretable expansion traces.

2. Dual-Resolution Recursive Energy Model

At each decision time t, the low-level controller receives a decision context x_t, consisting of the observation history, the current partial observation, and, when available, the current high-level subgoal. The controller selects a primitive action $a \in \mathcal{A}$. Figure 1 summarizes how the proposed model unifies contracted System 1-style evaluation and expanded System 2-style reasoning within the same recursive energy tree.

The model scores each candidate action through a compo-

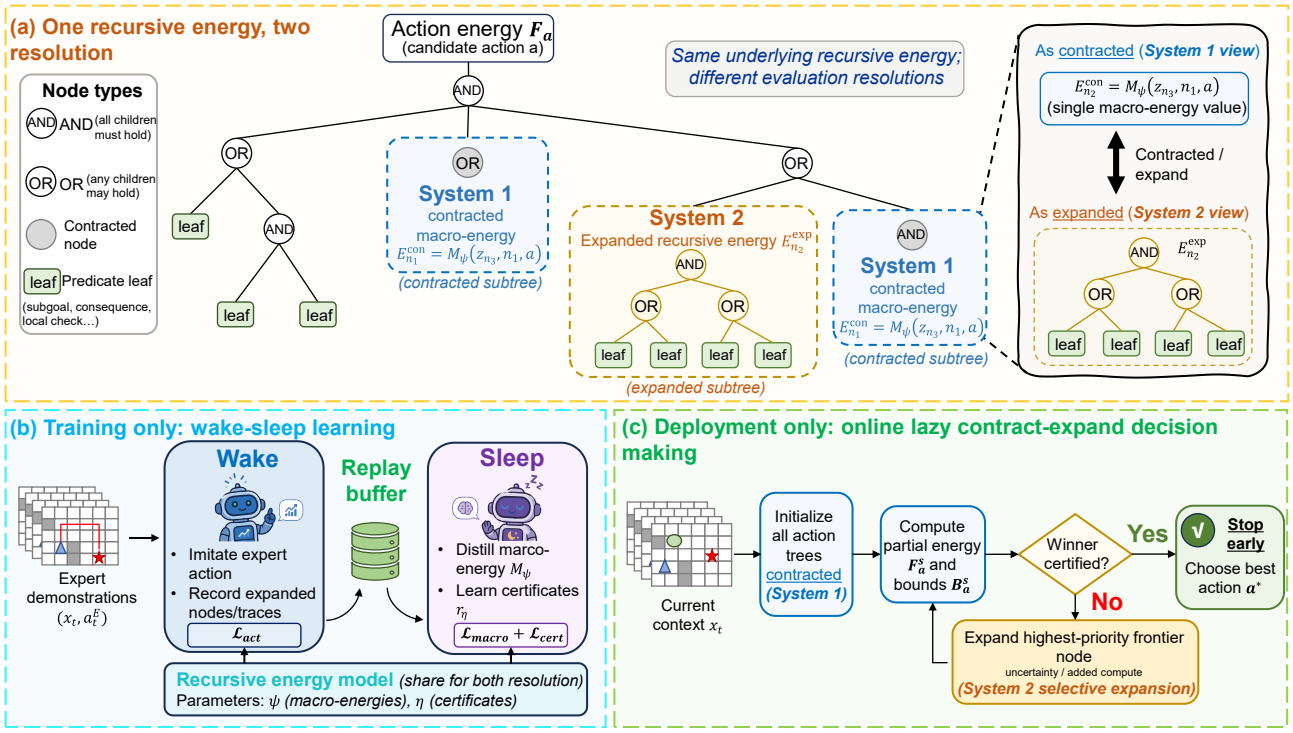


Figure 1. Overview of the proposed framework. (a) Each candidate action is scored by a recursive energy over an action-conditioned proof-program tree. Internal subtrees may be contracted into learned macro-energies or expanded into lower-level predicates and composition nodes, yielding mixed-resolution evaluation. (b) During training, wake–sleep learning imitates expert actions, records expanded traces, distills macro-energies, and calibrates contraction-error certificates. (c) During deployment, the learned macro-energies and certificates are used for online contract–expand inference, which expands uncertain frontier nodes only until the winning action is certified or the compute budget is exhausted.

sitional support tree. Leaves represent grounded decision factors, such as local traversability, target visibility, object compatibility, or subgoal progress. Internal nodes combine these factors through differentiable AND/OR operations, and the root energy defines the action score. Thus, action values are not learned as independent state-action mappings, but are assembled from reusable predicate-level factors and action-conditioned subtrees. The same tree admits three evaluation modes: a contracted evaluation using learned macro-energies, a fully expanded recursive evaluation over explicit predicates, and a mixed-resolution evaluation that expands only selected subtrees. This section defines the action tree, the recursive energy, and the induced action policy. Section 3 then describes how macro-energies and certificates are learned and used for online contract–expand inference.

2.1. Action-Conditioned Proof-Program Trees

Let $z_t = q_\phi(x_t)$ be an encoded decision context. For each action $a \in \mathcal{A}$, we define an action-conditioned proof-program tree

$$T_a = (\mathcal{V}_a, r_a, \mathrm{ch}, \mathrm{op}),$$

where $\mathcal{V}_a$ is the node set, r_a is the root, $\mathrm{ch}(n)$ denotes the children of node n, and $\mathrm{op}(n) \in \{\wedge, \vee\}$ is the composition type of an internal node. We denote the leaf and internal nodes by $\mathcal{L}_a$ and $\mathcal{I}_a$.

Leaves correspond to grounded action-relevant checks, including subgoal conditions, local executability predicates, object or relation predicates, and short-horizon action consequences. Internal nodes combine these checks into higher-level support for the action. The tree structure may be specified from task semantics or instantiated from action/subgoal templates; learning assigns energies to leaves, learns reusable macro-energies for subtrees, and determines how much of the tree to evaluate.

When action-conditioned lookahead is available, the model also computes a consequence representation u^a_t:

$$u^a_t = \rho\big(\xi^a_{t,1}, \ldots, \xi^a_{t,M}\big), \qquad \xi^a_{t,m} \sim p_\omega(\xi \mid z_t, a).$$

The source of u^a_t may be a learned transition model, simulator, or programmatic transition function. In domains without lookahead, this term can be omitted.

2.2. Recursive Energies at Two Resolutions

Leaf energies. Each leaf $\ell \in \mathcal{L}_a$ is scored by a predicate network. Let e_ℓ and e_a denote leaf and action embeddings. The model predicts

$$s_\ell(z_t, u_t^a, a) = \sigma(f_\theta(z_t, u_t^a, e_\ell, e_a)),$$

and converts this soft truth value into an energy

$$\epsilon_\ell(z_t, u_t^a, a) = -\log s_\ell(z_t, u_t^a, a).$$

For negated predicates, we replace s_ℓ by $1 - s_\ell$. Low leaf energy indicates that the corresponding grounded condition supports action a under context x_t.

Expanded recursive energy. Expanded evaluation recursively scores a subtree by evaluating its children and composing their energies. We use

$$A_\wedge(x_1, \dots, x_m) = \sum_{i=1}^{m} x_i,$$
$$A_\vee(x_1, \dots, x_m) = -\tau \log \sum_{i=1}^{m} \exp(-x_i/\tau).$$

where $A_\wedge$ accumulates required support and $A_\vee$ is a soft minimum over alternatives.

The fully expanded energy of node n is

$$E_n^{\exp} = \begin{cases} \epsilon_n(z_t, u_t^a, a), & n \in \mathcal{L}_a, \\ c_n^{\exp} + A_{\mathrm{op}(n)}\left(\{E_c^{\exp} : c \in \mathrm{ch}(n)\}\right), & n \in \mathcal{I}_a, \end{cases}$$

where $c_n^{\exp}$ is the explicit evaluation cost of the subtree rooted at n. The fully expanded action score is

$$F_a^{\exp}(x_t) = E_{r_a}^{\exp}.$$

Contracted macro-energy. To enable fast evaluation, each internal node also has a learned macro-energy

$$E_n^{\mathrm{con}}(z_t, a) = M_\psi(z_t, e_n, e_a) + c_n^{\mathrm{con}},$$

where $c_n^{\mathrm{con}} < c_n^{\exp}$. This contracted value approximates the expanded energy of the same subtree; it is a compressed evaluation, not a separate policy.

Mixed-resolution partial energy. Let $S_a \subseteq \mathcal{I}_a$ be the set of internal nodes expanded for action a. The partial energy of node n under S_a is

$$E_n^{S_a} = \begin{cases} \epsilon_n(z_t, u_t^a, a), & n \in \mathcal{L}_a, \\ E_n^{\mathrm{con}}(z_t, a), & n \in \mathcal{I}_a,\ n \notin S_a, \\ c_n^{\exp} + A_{\mathrm{op}(n)}\big(\{E_c^{S_a}\}_{c \in \mathrm{ch}(n)}\big), & n \in \mathcal{I}_a,\ n \in S_a. \end{cases} \tag{1}$$

The resulting action score is

$$F_a^{S_a}(x_t) = E_{r_a}^{S_a}.$$

When $S_a = \varnothing$, the action is scored by contracted evaluation. When $S_a = \mathcal{I}_a$, it is fully expanded. Intermediate S_a values yield selective deliberation within the same recursive energy.

2.3. Energy-Based Action Policy

Given an action score $F_a(x_t)$ produced by any expansion set, the model defines

$$\pi_\Theta(a \mid x_t) = \frac{\exp(-F_a(x_t))}{\sum_{a' \in \mathcal{A}} \exp(-F_{a'}(x_t))},$$

where $\Theta = \{\phi, \theta, \psi, \omega\}$ denotes the scoring parameters. The predicted action is

$$\hat{a}_t = \arg\min_{a \in \mathcal{A}} F_a(x_t).$$

Thus, the recursive energy defines action scoring for any fixed resolution, while Section 3 defines how the model learns contracted evaluations and chooses expansion sets.

3. Learning and Online Lazy Contract–Expand Inference

We now describe the two procedures built on the recursive energy. During training, wake–sleep learning uses expert decision points (x_t, a_t^E) to learn action scoring, macro-energies, and contraction certificates. During deployment, the trained model uses those certificates for online contract–expand inference; wake–sleep is no longer run.

3.1. Wake–Sleep Learning

Wake learns from expert actions and records the expanded traces used by the current model. Sleep replays these traces to distill expanded subtree energies into contracted macro-energies and calibrate certificates.

Wake: imitation with trace collection. For each expert decision point (x_t, a_t^E), the current contract–expand evaluator produces expansion sets $\{S_{a,t}\}_{a \in \mathcal{A}}$ and scores $\{F_a^{S_{a,t}}(x_t)\}_{a \in \mathcal{A}}$. The primary wake loss is behavior cloning under these mixed-resolution scores:

$$\mathcal{L}_{\mathrm{act}} = -\log \frac{\exp(-F_{a_t^E}^{S_{a_t^E,t}}(x_t))}{\sum_{a \in \mathcal{A}} \exp(-F_a^{S_{a,t}}(x_t))}.$$

We use

$$\mathcal{L}_{\mathrm{wake}} = \mathcal{L}_{\mathrm{act}} + \lambda_{\mathrm{aux}} \mathcal{L}_{\mathrm{aux}},$$

where $\mathcal{L}_{\mathrm{aux}}$ contains implementation-specific stabilizers. In our experiments, these include an expanded-action loss

using $F_a^{\exp}$, optional leaf-level binary cross-entropy when predicate labels are available, and a compute penalty on expanded nodes.

After the wake update, the model stores the decision context and expansion traces and recomputes z_t and u_t^a during sleep:

$$\mathcal{B} \leftarrow \mathcal{B} \cup \left\{ \left(x_t,\ a_t^E,\ \{\xi_t^a\}_{a\in\mathcal{A}},\ \{S_{a,t}\}_{a\in\mathcal{A}}\right) \right\}.$$

These traces are model-generated expansion records, not assumed ground-truth expert reasoning.

Sleep: compression and certification. During sleep, the model samples replay tuples from $\mathcal{B}$. For each replayed expanded node $(t, a, n) \in \mathcal{R}$, we recompute the fully expanded subtree energy $E_n^{\exp}$ and use it as the target for the contracted macro-energy:

$$\mathcal{L}_{\text{macro}} = \sum_{(t,a,n)\in\mathcal{R}} \ell_{\text{macro}}\left(E_n^{\text{con}}(z_t, a), \text{sg}(E_n^{\exp}(z_t, u_t^a, a))\right),$$

where ℓ_{macro} is a regression loss and $\text{sg}(\cdot)$ stops gradients through the expanded target.

The certificate head $r_\eta(z_t, e_n, e_a) \geq 0$ estimates the contraction gap

$$\Delta_n = \left|E_n^{\exp}(z_t, u_t^a, a) - E_n^{\text{con}}(z_t, a)\right|.$$

We train certificates with the asymmetric calibration loss

$$\mathcal{L}_{\text{cert}} = \sum_{(t,a,n)\in\mathcal{R}} w_n \Big[\lambda_{\text{under}} \big(\text{sg}(\Delta_n) - r_n \big)_+ + \lambda_{\text{over}} \big(r_n - \text{sg}(\Delta_n) - \delta \big)_+ \Big]. \quad (2)$$

Here, $r_n = r_\eta(z_t, e_n, e_a)$, $(x)_+ = \max(x, 0)$, and w_n can upweight important nodes such as action roots. We set $\lambda_{\text{under}} > \lambda_{\text{over}}$ to penalize under-coverage more strongly than conservative over-coverage, with $\delta \geq 0$ denoting a slack tolerance. The sleep objective is

$$\mathcal{L}_{\text{sleep}} = \mathcal{L}_{\text{macro}} + \lambda_{\text{cert}} \mathcal{L}_{\text{cert}}.$$

Overall, the core training objective is

$$\mathcal{L} = \mathcal{L}_{\text{act}} + \mathcal{L}_{\text{macro}} + \mathcal{L}_{\text{cert}}. \quad (3)$$

Here, $\mathcal{L}_{\text{act}}$ imitates expert actions, $\mathcal{L}_{\text{macro}}$ distills expanded subtree evaluations into macro-energies, and $\mathcal{L}_{\text{cert}}$ calibrates contraction-error certificates. Optional grounding and compute regularization terms are added when available. Wake and sleep updates are alternated during training.

3.2. Online Lazy Contract–Expand Decision Making

At deployment, all action trees start contracted. The model then expands only enough structure to certify the current action choice or exhaust the compute budget.

Contraction-error bounds. For an internal node n, the learned certificate is used as an empirical bound on contraction error:

$$\left|E_n^{\exp}(z_t, u_t^a, a) - E_n^{\text{con}}(z_t, a)\right| \leq r_n, \quad r_n = r_\eta(z_t, e_n, e_a). \quad (4)$$

For a partial expansion set S_a, node-level uncertainty is propagated upward as

$$B_n^{S_a} = \begin{cases} 0, & n \in \mathcal{L}_a, \\ r_n, & n \in \mathcal{I}_a,\ n \notin S_a, \\ \sum_{c\in\text{ch}(n)} B_c^{S_a}, & n \in S_a,\ \text{op}(n) = \wedge, \\ \max_{c\in\text{ch}(n)} B_c^{S_a}, & n \in S_a,\ \text{op}(n) = \vee. \end{cases}$$

The root bound $B_a^{S_a} = B_{r_a}^{S_a}$ estimates the uncertainty in the current partial score $F_a^{S_a}(x_t)$.

Winner certification. Let

$$a^\star = \arg\min_{a\in\mathcal{A}} F_a^{S_a}(x_t)$$

be the current best action. We certify $a^\star$ if, for every $a \neq a^\star$,

$$F_a^{S_a}(x_t) - F_{a^\star}^{S_{a^\star}}(x_t) > B_a^{S_a} + B_{a^\star}^{S_{a^\star}}. \quad (5)$$

Under calibrated certificates, this margin implies that further expansion cannot change the winning action. If Eq. (5) holds, inference stops and returns $a^\star$.

Selective expansion. If the winner is not certified, the model expands frontier nodes with high uncertainty per added compute:

$$\text{priority}_a(n) = \frac{B_n^{S_a}}{c_n^{\exp} - c_n^{\text{con}} + \varepsilon}.$$

The selected node is added to S_a, its children become available for evaluation, and the affected scores and bounds are recomputed. The process repeats until certification, budget exhaustion, or no expandable frontier remains. The final action is

$$\hat{a}_t = \arg\min_{a\in\mathcal{A}} F_a^{S_a^\star}(x_t),$$

where $S_a^\star$ is the final expansion set.

This inference rule makes deliberation adaptive: contracted evaluation is used when certified, and expansion is invoked only for subtrees that may affect the action decision.

4. Experiments

4.1. Environment and Task Formulation

We evaluate whether the proposed dual-resolution recursive energy model supports sample-efficient, compositional,

Algorithm 1 Deployment-Time Online Contract–Expand Decision Making

Require: decision context x_t, action trees $\{T_a\}_{a\in\mathcal{A}}$, compute budget K

Encode context $z_t = q_\phi(x_t)$
Initialize $S_a \leftarrow \varnothing$ for all $a \in \mathcal{A}$
for $k = 1, \dots, K$ **do**
  Compute $F_a^{S_a}(x_t)$ and $B_a^{S_a}$ for all actions
  Let $a^\star \leftarrow \arg\min_a F_a^{S_a}(x_t)$
  if $a^\star$ satisfies Eq. (5) **then**
    break
  end if
  Select the highest-priority expandable pair (a, n)
  Update $S_a \leftarrow S_a \cup \{n\}$
end for
return $\hat{a}_t = \arg\min_a F_a^{S_a}(x_t)$

and compute-adaptive control in embodied symbolic environments. We use two benchmarks: BabyAI (Chevalier-Boisvert et al., 2019), a language-conditioned gridworld benchmark with primitive navigation and object-interaction actions, and MiniHack (Samvelyan et al., 2021), a NetHack-based benchmark with command-driven, inventory-mediated, and longer-horizon symbolic interactions. For BabyAI, we use six environments spanning navigation, door interaction, unlocking, obstacle clearing, and target pickup: `GoToObj-v0`, `OpenDoor-v0`, `PickupLoc-v0`, `UnlockPickup-v0`, `UnblockPickup-v0`, and `BlockedUnlockPickup-v0`. For MiniHack, tasks cover compact and large-room navigation, eating, wearing equipment, praying, levitation-based lava crossing, and tool-mediated dungeon interaction.

We focus on subgoal-conditioned low-level control rather than end-to-end high-level task decomposition. Given the current subgoal and partial local observation, the controller selects the next primitive action. This setting isolates local decision-making: routine states can be handled by contracted habitual evaluation, while ambiguous states such as search, doorway traversal, obstacle clearing, or command/inventory interaction may trigger selective expansion.

4.2. Success Rate and Sample Efficiency

We compare our method with same-input imitation baselines, including Flat-BC and H-BC, as well as PPO-based RL baselines. All baselines use the same subgoal-conditioned observation interface as our method, so the comparison isolates the effect of recursive energy composition and lazy contract–expand inference rather than differences in task conditioning. Implementation details are provided in Appendix E.

Table 1 reports Success@100 on BabyAI and MiniHack,

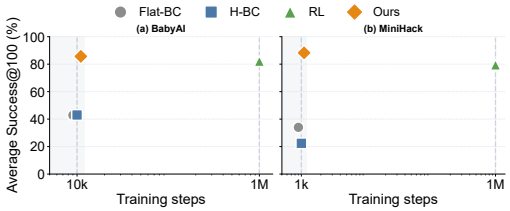


Figure 2. Sample efficiency comparison on BabyAI and MiniHack. The x-axis reports the amount of training experience, and the y-axis reports average Success@100 across environments. BC-based methods use expert demonstration steps, while PPO uses online environment interaction steps.

and Figure 2 summarizes average success versus training experience. BabyAI imitation methods use 400 demonstrations per environment ($\sim 10^4$ expert steps), and MiniHack imitation methods use 40 demonstrations ($\sim 10^3$ expert steps), whereas PPO baselines use 1M online interaction steps. Since demonstrations and online interaction are different forms of supervision, we interpret this comparison as a sample-efficiency profile rather than a direct step-for-step equivalence.

On simple navigation or direct-interaction tasks, our method is competitive with the strongest baselines. Its advantage becomes clearer on structured interaction tasks that require composing several local skills. On `UnblockPickup`, the agent must coordinate target search, temporary obstacle manipulation, and final pickup; our method reaches 86% success, compared with 15% for Flat-BC and 18% for H-BC. On `BlockedUnlockPickup`, where key acquisition, obstacle clearing, unlocking, door traversal, and final pickup must be coordinated, our method reaches 60%, while both same-input imitation baselines obtain 0%. Averaged across tasks, our method reaches 85.7% on BabyAI and 88.3% on MiniHack, compared with 82.2% and 79.5% for PPO under a much larger online interaction budget.

These results suggest that recursive energy structure provides a useful compositional inductive bias for sample-efficient low-level control. Rather than learning each state-action mapping independently, the model assembles action scores from reusable grounded decision factors and action-conditioned subtrees, which is especially beneficial when success depends on composing multiple local skills.

4.3. Compositional Transfer to Bosslevel

We further evaluate the BabyAI-trained model on `BossLevel-v0` missions that match the skill family covered by our training environments, without task-specific retraining. These missions compose navigation, door interaction, unlocking, obstacle clearing, and pickup into longer execution chains. This setting therefore tests whether the learned low-level decision factors can be reused under new

Table 1. Success rates on BabyAI and MiniHack environments.

(a) BabyAI

Environment	Flat-BC	H-BC	RL	Ours
GoToObj	78%	96%	**100%**	98%
OpenDoor	82%	61%	87%	**93%**
PickupLoc	82%	83%	89%	**91%**
UnlockPickup	0%	0%	**96%**	86%
UnblockPickup	15%	18%	66%	**86%**
BlockedUnlockPickup	0%	0%	55%	**60%**

(b) MiniHack

Environment	Flat-BC	H-BC	RL	Ours
Room-5x5	93%	85%	98%	**100%**
Room-15x15	**48%**	8%	39%	41%
Eat	4%	0%	**100%**	**100%**
Wear	54%	44%	**100%**	98%
Pray	12%	0%	**100%**	93%
LavaCross-Ring	56%	0%	**100%**	88%
LavaCross-Potion	5%	42%	**99%**	87%
WoD-Medium	0%	0%	0%	**99%**

compositions of familiar skills.

The model achieves **47%** Success@100 on BossLevel-v0. While long-horizon composed missions remain challenging, this result suggests preliminary compositional transfer within the covered skill family: the controller can reuse predicate-level decision factors learned from simpler source tasks, while richer replanning and recovery may further improve robustness.

4.4. Mechanism-Isolation Ablations

We add mechanism-isolation ablations to clarify where the gains of our method come from: action-level supervision, semantic leaf grounding, recursive energy aggregation, and certificate-guided lazy inference. These ablations are designed to address whether the improvement mainly comes from hand-designed predicates, the recursive energy tree, or the adaptive expand–contract inference mechanism. Table 2 summarizes the results.

Action-level learning is necessary. Removing the action-level loss causes complete failure across all tested environments, even though leaf-level supervision is still retained. This shows that grounded predicates alone do not define an executable control policy. The model must learn how predicate energies compose into primitive action preferences; otherwise, semantic predicates remain descriptive but cannot drive closed-loop behavior.

Leaf grounding provides semantic and sample-efficiency bias. Removing leaf supervision substantially weakens performance under the default demonstration budget, especially on structured interaction tasks. The drop is small on OpenDoor but severe on PickupLoc and UnblockPickup, where success decreases from 91% to 14% and from 84% to 12%, respectively. This indicates that leaf supervision provides important semantic grounding for reusable local decision factors. However, when the demonstration budget is increased from 400 to 2000 trajectories, the w/o-leaf variant recovers to 89% on PickupLoc and 75% on UnblockPickup. Thus, manual leaf grounding should be viewed primarily as a source of semantic regularization and sample efficiency, rather than as the only possible source of policy competence. These results also suggest a natural path toward more data-driven variants, such as learned predicate discovery or learned tree construction, which could reduce manual engineering at the cost of additional data.

Certificate-guided lazy inference matters beyond compute reduction. We isolate the lazy inference mechanism with a **w/o lazy inference** ablation. This ablation keeps the same trained recursive tree and matches the average expansion budget of lazy inference, but removes certificate-guided adaptive expand–contract decisions. Instead, it uses a non-adaptive fixed-budget expansion rule, expanding nodes according to a fixed-depth/random order until the average number of expanded nodes matches that of lazy inference. Because the compute budget and tree access are controlled, this ablation tests whether the benefit comes from adaptive allocation of expansion rather than merely from evaluating a comparable number of nodes.

The w/o-lazy variant suffers modest-to-moderate degradation on the simpler tasks and a much larger drop on UnblockPickup, where success decreases from 84% to 59%. This 25-point drop indicates that, in more compositional tasks, it is not enough to expand a fixed or random subset of the tree. The certificates are needed to decide when a subtree can be safely contracted and when additional reasoning should be allocated to decision-critical factors such as obstacle clearing, navigation, and pickup.

Recursive energy aggregation improves compositional generalization. We isolate the recursive energy tree with a **w/o recursive tree** ablation. This model keeps the same predicate predictions and the current latent state z_t, but replaces recursive energy aggregation with an MLP action head over leaf logits/probabilities and z_t. Thus, it preserves the predicate interface while removing the tree-structured energy prior.

The flat Predicate-MLP remains competitive, and even performs better, on some locally reactive tasks such as OpenDoor and PickupLoc. This is expected: when the correct action can be inferred from local predicate-action correlations, a direct MLP head can fit the action labels well. However, the same model drops sharply on UnblockPickup, from 84% to 24%. This suggests that

predicate features alone are insufficient for the most compositional task. The recursive tree structure provides an inductive bias for composing search, navigation, obstacle-clearing, and pickup factors into action-level energies, which becomes important when local reactive correlations no longer generalize.

Taken together, these ablations separate the roles of the main components. Leaf grounding supplies semantic alignment and sample efficiency; the action loss turns grounded predicates into executable action preferences; certificate-guided lazy inference allocates computation to the relevant parts of the tree; and recursive energy aggregation provides the compositional structure needed for more complex interaction tasks.

Table 2. Mechanism-isolation ablations.

Environment	Full	w/o action	w/o leaf	w/o lazy	w/o tree
`GoToObj`	98%	0%	88%	84%	79%
`OpenDoor`	93%	0%	94%	88%	99%
`PickupLoc`	91%	0%	14%	89%	99%
`UnblockPickup`	86%	0%	12%	59%	24%

w/o lazy inference removes certificate-guided adaptive expansion and uses a non-adaptive fixed-depth/fixed-budget rule with a matched average budget. **w/o recursive tree** keeps the same predicate outputs and latent state z_t, but replaces recursive energy aggregation with an MLP action head. The w/o-leaf results show that manual leaf grounding improves semantic regularization and sample efficiency.

4.5. Online Lazy Contract–Expand Inference and Contraction Reliability

Lazy versus full expansion. We next evaluate whether lazy contract–expand inference can reduce explicit recursive computation while preserving task success. This comparison tests the central role of learned contraction: if macro-energies provide reliable compressed evaluations, the controller should not need to fully expand every action tree at every decision point.

Figure 3 compares lazy inference with fully expanded evaluation over three runs. Across BabyAI environments, lazy inference reduces the number of expanded nodes by 44.6% on average, from 35.00 to 19.99 nodes, while average success changes only from 86.6% under full expansion to 85.3% under lazy inference. Thus, learned contraction preserves most of the benefit of expanded reasoning while avoiding nearly half of the explicit node evaluations.

Lazy inference is certificate-guided rather than uniformly compressed: routine states can remain contracted, while uncertain action rankings trigger recursive expansion before acting. This yields a computation–reliability tradeoff in which the controller saves node evaluations when possible, but retains a mechanism for escalating uncertain decisions.

Qualitative expansion patterns. We provide qualitative examples to examine how lazy contract–expand inference

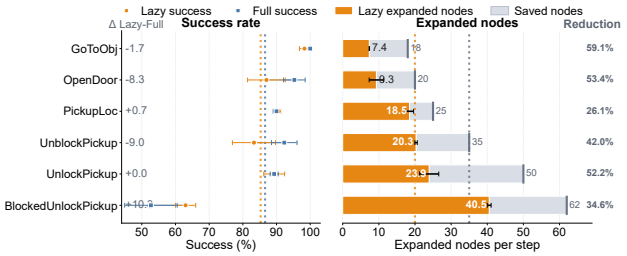


Figure 3. Lazy contract–expand inference versus fully expanded evaluation on BabyAI over multiple runs. Left: paired success comparison between lazy and full expansion. Right: expanded-node usage, where the orange portion denotes nodes expanded by lazy inference and the gray portion denotes nodes saved relative to full expansion.

allocates computation across decision contexts. Figure 4 shows both a rollout-level trace and local reasoning-tree views in BabyAI. At the rollout level, expansion is highly non-uniform across the trajectory. Routine movement within a room or during locally unambiguous navigation is often handled with little expansion, suggesting that contracted macro-energies are sufficient for familiar transitions. In contrast, states involving doorway traversal, target search, object pickup, or obstacle interaction trigger deeper recursive evaluation, indicating that the controller allocates additional computation to decision-critical stages.

The local views further show that expansion is state-dependent rather than tied to a fixed action label. In all highlighted states, the selected primitive action is `forward`; however, the reasoning tree is expanded to different extents. In familiar states, the tree remains mostly contracted because the macro-evaluation already gives a reliable preference. In moderately ambiguous states, only part of the tree is unfolded, yielding a mixed-resolution decision. In more difficult states, such as those involving search, doorway continuation, or nearby interaction constraints, the controller expands more of the tree before committing to the same `forward` action. These examples support the intended dual-resolution behavior: contracted macro-energies provide reusable fast evaluations, while expanded reasoning supplies additional deliberation when the local context requires it.

Learning progression. Figure 5 illustrates how contraction becomes reliable during wake–sleep training. On the same evaluation case, early checkpoints expand many nodes but still fail, indicating that additional computation alone is insufficient before the underlying action energies are learned. At intermediate epochs, fully expanded evaluation succeeds before lazy inference, suggesting that explicit recursive action evaluation is learned before its contracted approximation becomes reliable. By the final epoch, lazy inference also succeeds while using fewer expanded nodes and fewer

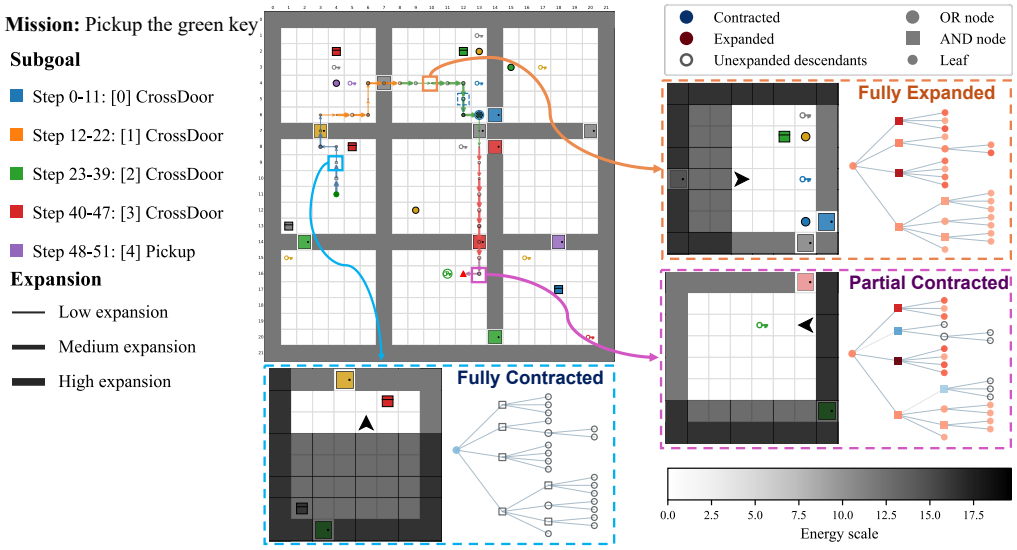


Figure 4. Qualitative BabyAI rollout and adaptive expansion for the same primitive action `forward`. Arrow thickness indicates the amount of lazy recursive expansion. Filled red nodes denote expanded nodes or leaves, and blue nodes denote contracted nodes.

environment steps.

This progression supports the dual-resolution interpretation. Expanded reasoning first provides the detailed decision computation, and sleep training later compresses useful recursive evaluations into contracted macro-energies. The trace also exposes a recognizable composed routine: pick up the key, temporarily drop it to clear an obstacle, recover the key, unlock/open the door, discard the key after use, cross into the next room, and pick up the target object. Thus, lazy contract–expand inference provides not only adaptive computation, but also an interpretable record of when the controller relies on compressed evaluation and when it invokes deeper reasoning. Additional experiments and results are provided in Appendix B.

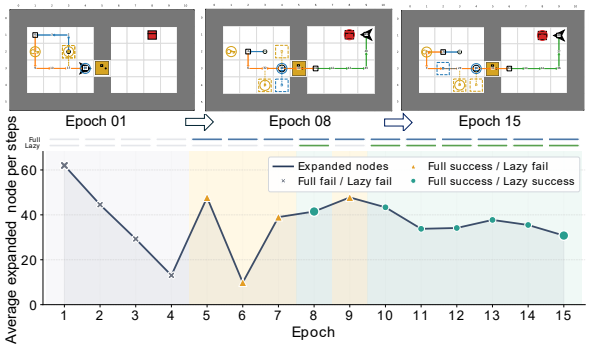


Figure 5. Learning progression on the same evaluation case across training epochs. The agent gradually learns a key-and-obstacle routine: pick up the key, temporarily drop it to clear an obstacle, recover the key, unlock/open the door, discard the key after use, cross into the next room, and pick up the target object. At later epochs, lazy inference succeeds with fewer expanded nodes and fewer environment steps, indicating that expanded reasoning has been compressed into reliable contracted evaluations.

5. Conclusion

We introduced a wake–sleep dual-resolution recursive energy framework for subgoal-conditioned sequential decision-making. The framework scores primitive actions through action-conditioned reasoning trees, where contracted macro-energies support fast habitual responses and expanded evaluations provide slower deliberative reasoning when needed. Wake–sleep training learns grounded leaf-level decision factors, distills expanded subtree reasoning into contracted macro-energies, and calibrates certificates for lazy contract–expand inference. Experiments on BabyAI and MiniHack show improved performance over same-input imitation baselines on structured interaction tasks, competitive results with PPO under a smaller supervision budget, and preliminary BossLevel transfer through reuse of familiar low-level skills. These results support selective recursive expansion as a practical mechanism for sample-efficient, compute-adaptive, and interpretable low-level control.

By exposing which predicates support an action and when additional computation is invoked, the framework makes local decision-making more traceable. The current instantiation uses programmatically grounded predicates and short imagined futures, which provide a controlled foundation for evaluating the core dual-resolution mechanism. Extending the framework to learned predicate grounding, learned world models, richer high-level replanning, and RL-based fine-tuning is a natural next step that could broaden its applicability to more open-ended embodied domains.

References

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Banino, A., Balaguer, J., and Blundell, C. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Belanger, D. and McCallum, A. Structured prediction energy networks. In *International Conference on Machine Learning*, pp. 983–992. PMLR, 2016.

Belanger, D., Yang, B., and McCallum, A. End-to-end learning for structured prediction energy networks. In *International Conference on Machine Learning*, pp. 429–439. PMLR, 2017.

Boney, R., Kannala, J., and Ilin, A. Regularizing model-based planning with energy-based models. In *Conference on Robot Learning*, pp. 182–191. PMLR, 2020.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019.

Daw, N. D., Niv, Y., and Dayan, P. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8 (12):1704–1711, 2005.

Dolan, R. J. and Dayan, P. Goals and habits in the brain. *Neuron*, 80(2):312–325, 2013.

Dragan, A. D., Lee, K. C., and Srinivasa, S. S. Legibility and predictability of robot motion. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 301–308. IEEE, 2013.

Evans, J. S. B. In two minds: dual-process accounts of reasoning. *Trends in cognitive sciences*, 7(10):454–459, 2003.

Evans, J. S. B. and Stanovich, K. E. Dual-process theories of higher cognition: Advancing the debate. *Perspectives on psychological science*, 8(3):223–241, 2013.

Ferguson, N. Capturing human behaviour. *Nature*, 446 (7137):733–733, 2007.

Gladstone, A., Nanduru, G., Islam, M. M., Han, P., Ha, H., Chadha, A., Du, Y., Ji, H., Li, J., and Iqbal, T. Energy-based transformers are scalable learners and thinkers. *arXiv preprint arXiv:2507.02092*, 2025.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. The Wake-Sleep algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

Janner, M., Li, Q., and Levine, S. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, volume 34, pp. 1273–1286, 2021.

Kahneman, D. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, NY, 2011. ISBN 9780374275631.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., Huang, F., et al. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Lee, S. W., Shimojo, S., and O'doherty, J. P. Neural computations underlying arbitration between model-based and model-free learning. *Neuron*, 81(3):687–699, 2014.

Liu, M., He, T., Xu, M., and Zhang, W. Energy-based imitation learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 809–817, 2021.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

Marsch, L. A. Digital health data-driven approaches to understand human behavior. *Neuropsychopharmacology*, 46(1):191–196, 2021.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Küttler, H., Grefenstette, E., and Rocktäschel, T. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

Shafiullah, N. M., Cui, Z., Altanzaya, A. A., and Pinto, L. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35: 22955–22968, 2022.

Sloman, S. A. The empirical case for two systems of reasoning. *Psychological bulletin*, 119(1):3, 1996.

Stanovich, K. E. and West, R. F. Advancing the rationality debate. *Behavioral and brain sciences*, 23(5):701–717, 2000.

Sun, R., Slusarz, P., and Terry, C. The interaction of the explicit and the implicit in skill learning: a dual-process approach. *Psychological review*, 112(1):159, 2005.

Teerapittayanon, S., McDanel, B., and Kung, H.-T. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

A. Related Work

Human behavior modeling Human behavior modeling is important in domains such as public health and human–robot collaboration, where prediction alone is often insufficient without interpretable decision structure (Ferguson, 2007; Marsch, 2021; Dragan et al., 2013). Probabilistic and deep models can capture event dynamics, trajectories, or action sequences, but often provide limited insight into whether behavior arises from habit, planning, or deliberation. Dual-process theories distinguish fast habitual responses from slower deliberative reasoning, often described as System 1 and System 2 (Kahneman, 2011; Evans, 2003; Stanovich & West, 2000). Recent energy-based perspectives further suggest that energy models can support verification and additional inference-time computation, making them a natural substrate for deeper "thinking" beyond a single feed-forward prediction (Gladstone et al., 2025). Our framework connects these views by treating System 1 and System 2 as contracted and expanded evaluations of the same recursive energy: wake exposes useful expanded reasoning traces, while sleep consolidates them into macro-energies and calibrated certificates. Thus, habits are not a separate policy class, but compressed reasoning.

Energy-based and structured decision models. Energy-based models provide a general framework for assigning low energy to compatible configurations and high energy to incompatible ones, and naturally induce Gibbs distributions over actions or latent variables(LeCun et al., 2006). They have been used as flexible scoring models for perception, structured prediction(Belanger & McCallum, 2016; Belanger et al., 2017), planning, and decision-making(Boney et al., 2020). However, existing energy-based decision models typically define energies over candidate labels, state–action pairs, plans, trajectories, or latent variables (Liu et al., 2021), rather than over the recursive reasoning process that supports an action decision. Our work instead defines energy recursively over an action-conditioned reasoning tree. This makes the energy itself structured: action scores are supported by grounded leaf predicates and composed through AND/OR subtrees. As a result, the model can expose not only which action is preferred, but also which local reasoning factors support that preference.

Structured reasoning and adaptive deliberation. A broad line of work studies structured reasoning through symbolic programs, neuro-symbolic models, tree search, and explicit multi-step inference (Manhaeve et al., 2018; Yao et al., 2023). These methods expose more interpretable intermediate structure than black-box policies, but the decision of how much structure to evaluate is often fixed by the inference procedure or controlled by an external search or prompting strategy. Separately, adaptive computation methods allocate different amounts of neural computation to different inputs through learned halting or early-exit mechanisms (Graves, 2016; Teerapittayanon et al., 2016). Our framework combines these two ideas in a single object: reasoning depth is not an external module-level decision, but a resolution choice inside the same recursive energy. Lazy contract–expand inference expands only the parts of the tree needed to certify the current action decision, rather than uniformly evaluating the full reasoning structure.

B. Additional Qualitative and Diagnostic Results

B.1. Additional Case Studies

We provide an additional MiniHack case study to complement the BabyAI examples shown in the main text and to illustrate how lazy contract–expand inference behaves in a command- and inventory-mediated setting.

Figure 6 shows a successful rollout on `MiniHack-WoD-Medium-Full-v0`. The trajectory includes an initial pickup stage, a long movement segment, a command/inventory interaction stage, and a final descent stage. Most straightforward `move_east` actions are handled with no expansion, indicating that contracted macro-energies are sufficient for routine corridor navigation. In contrast, interaction-related steps such as `pickup` and `select_item` trigger light expansion, and the transition following the zap/select interaction triggers substantially deeper expansion. After this decision-critical transition, the remaining descent-directed movement again becomes mostly contracted. This MiniHack example highlights that the controller does not expand simply because the environment is more complex. Instead, expansion is allocated locally: routine navigation is treated habitually, while command, inventory, and transition-sensitive states invoke additional recursive evaluation. This supports the dual-resolution interpretation that contracted energies capture reusable action patterns, while expanded evaluation is reserved for consequential or uncertain decisions.

B.2. Contraction–Success Sweep

We further sweep the inference-time contraction setting on `PickupLoc-v0` to examine how success changes as the amount of explicit recursive reasoning varies. This experiment complements the main lazy-versus-full comparison by showing a

continuous tradeoff between computation and reliability, rather than only comparing two endpoints. For each setting, we measure the average number of expanded nodes, rollout success, and average episode length.

As shown in Figure 7, aggressive contraction expands only a few nodes and yields low success. In this regime, the controller relies heavily on contracted macro-energies, and unreliable macro-evaluations can dominate the action decision. As the expansion budget increases, success rises sharply and the average episode length decreases, indicating that additional recursive evaluation helps the model avoid early wrong actions and complete the task more efficiently. The improvement in episode length is important: the model does not merely succeed more often, but also reaches the goal using shorter and more stable behavior.

The best lazy setting reaches 91% success with about 19–21 expanded nodes on average, compared with 89% under full expansion with 25 nodes. We do not interpret this as evidence that partial expansion is universally better than full expansion. Rather, the result suggests that calibrated contraction can sometimes match full expansion while avoiding unnecessary evaluation of subtrees that do not affect the final action choice. Conversely, the low-success regime under very small expansion budgets shows that over-contraction is harmful when certificates and macro-energies are not sufficient to certify the winning action. Overall, the sweep supports the intended role of lazy inference: it should not minimize expansion blindly, but should expand enough of the tree to make a reliable action decision.

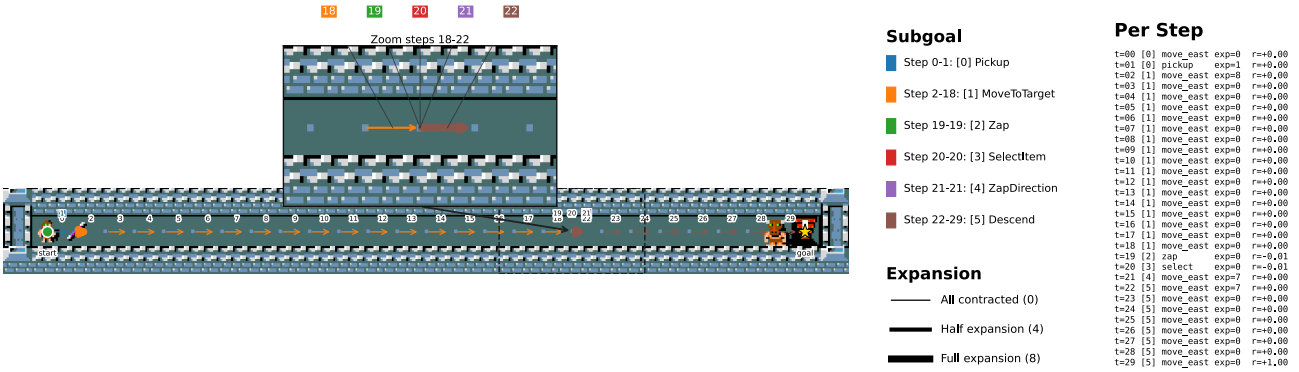


Figure 6. Qualitative rollout of MiniHack. Colors indicate high-level subgoals, and arrow thickness indicates the amount of reasoning expansion used by lazy inference.

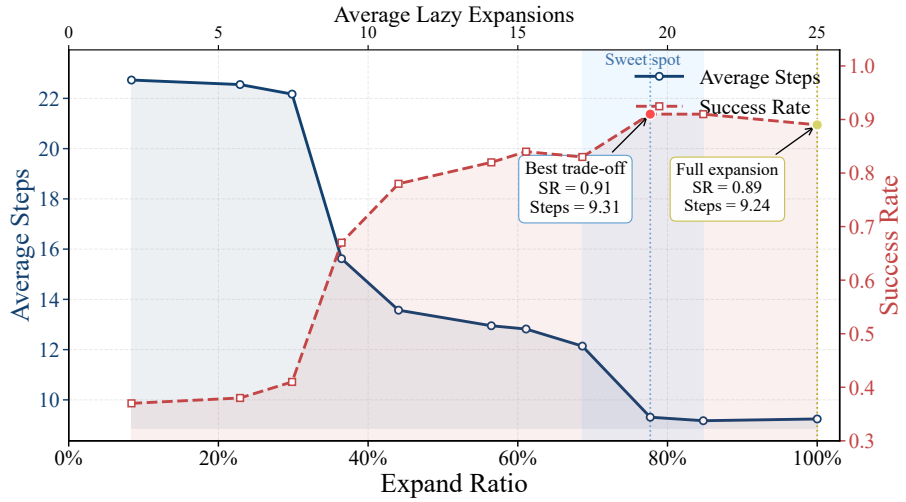


Figure 7. Contraction–success tradeoff on `PickupLoc-v0`. We vary the inference-time contraction setting and measure the average number of expanded nodes, rollout success, and average episode length. Excessive contraction uses very few expansions but leads to low success and longer rollouts. As more reasoning is allowed, success improves and the number of steps decreases. Lazy inference reaches its best success before full expansion in this sweep, while full expansion remains competitive.

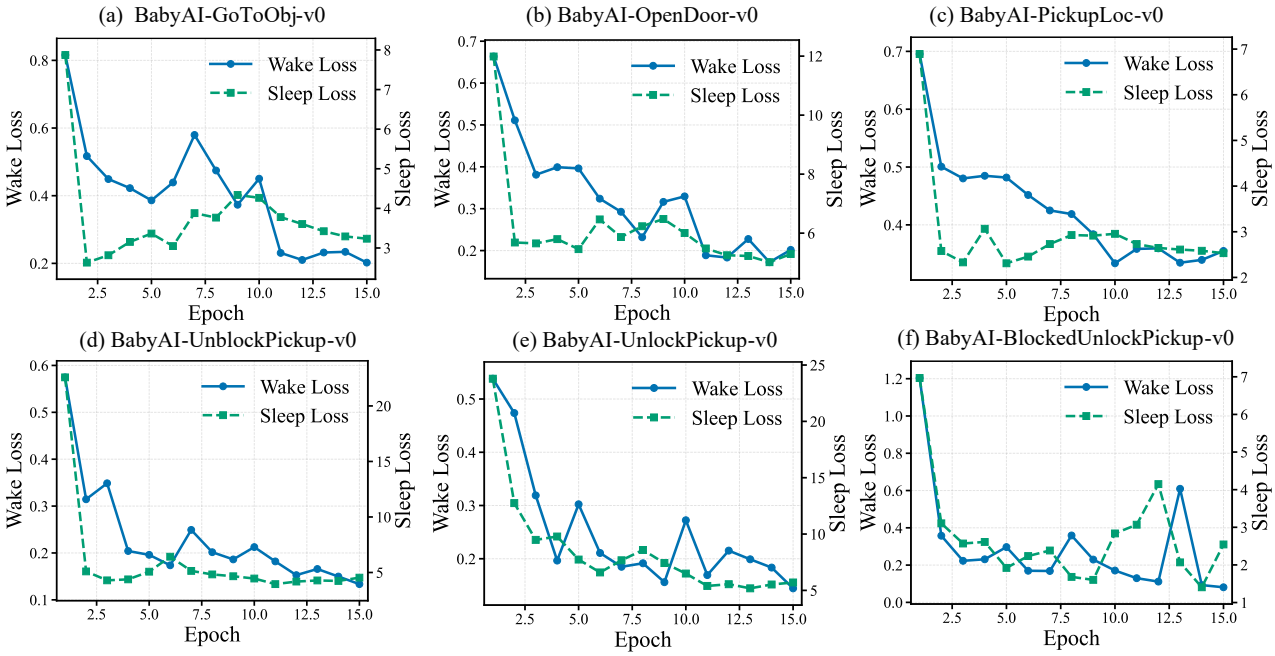


Figure 8. Training losses across epochs on the BabyAI environments. Wake loss measures the supervised decision objective during wake, while sleep loss measures macro-energy distillation and certificate calibration.

B.3. Training Dynamics and Expansion Diagnostics

Figure 8 and Figure 9 show wake–sleep training losses on BabyAI and MiniHack, respectively. Wake loss measures the supervised decision objective during wake, while sleep loss measures macro-energy distillation and certificate calibration. Across both benchmarks, the losses generally decrease and stabilize, suggesting that the model learns both explicit action evaluation and contracted macro-evaluations in a stable manner.

Figure 10 and Figure 11 report expansion diagnostics on BabyAI and MiniHack. The expansion ratio measures the fraction of internal reasoning nodes expanded during lazy inference. The certificate violation rate is the fraction of checked contracted internal nodes for which the learned certificate is smaller than the empirical gap between the contracted estimate and the fully expanded subtree energy. This metric therefore measures the reliability of learned contraction bounds, not direct disagreement with the fully expanded action decision. Across training, the model generally reduces explicit expansion while keeping certificate under-coverage low.

B.4. Over-Contraction Failure Case

Figure 12 shows a representative failure mode under an overly aggressive contraction setting. In this run, the sleep stage pushes the model to rely almost entirely on contracted macro-energies: the expand ratio approaches zero by the final epoch. However, the certificate violation rate against fully expanded evaluation increases sharply, indicating that the learned certificates no longer reliably cover the error between contracted and expanded subtree energies.

This case illustrates why minimizing expansion alone is not the goal of lazy inference. If contraction is too aggressive, the controller may avoid expanding subtrees whose explicit evaluation would change the action preference. In that regime, macro-energies can become overconfident approximations, and certificates can underestimate the true contraction error. The resulting policy may appear computationally efficient, but its decisions are no longer reliably aligned with the expanded recursive energy.

The failure case motivates the calibrated contract–expand design used in the main experiments. A useful lazy controller should contract routine or low-uncertainty subtrees, but still expand decision-critical regions when the current action ranking is not sufficiently certified. Thus, expansion reduction and certificate reliability must be considered together.

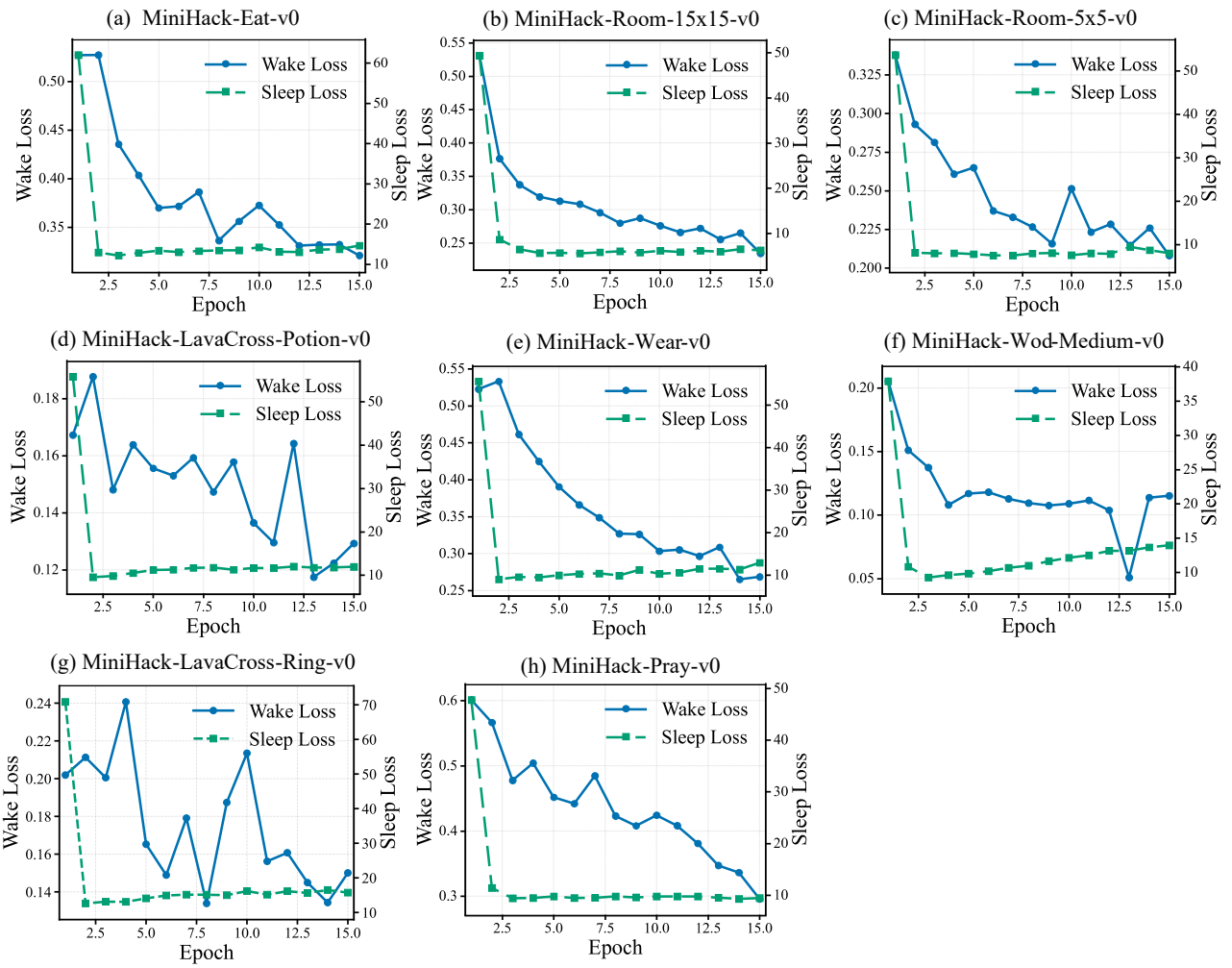


Figure 9. Training losses across epochs on the MiniHack environments. Wake loss measures the supervised decision objective during wake, while sleep loss measures macro-energy distillation and certificate calibration.

B.5. Temperature Sensitivity of Energy Aggregation

We also evaluate the sensitivity of the log-sum-exp energy aggregation to the temperature parameter τ. Since overly small or overly large temperatures may change the sharpness of the aggregation, we test $\tau \in \{0.25, 0.5, 1.0\}$ while keeping all other settings fixed. The default value used in our main experiments is $\tau = 0.5$.

Table 3 shows that performance is stable over this range. The default $\tau = 0.5$ is either the best or close to the best value across all tested environments. On `UnblockPickup`, the most compositional task in this ablation, success remains within a narrow range from 82% to 84%. On simpler tasks, the variation is also modest, with `GoToObj` staying between 97% and 98%, and `OpenDoor` between 90% and 93%. `PickupLoc` shows a slightly larger change when $\tau = 1.0$, but still remains within 86–91%. These results suggest that the method is not highly sensitive to the exact temperature choice, and that $\tau = 0.5$ provides a stable default for the log-sum-exp aggregation.

C. Example Action-Conditioned Recursive Energy Tree

This appendix illustrates one representative action-conditioned recursive energy tree used in our BabyAI experiments. We use an `UnblockPickup` instance as the running example because it contains several common forms of compositional interaction: target search, navigation, temporary obstacle clearing, safe dropping, doorway traversal, and final object pickup.

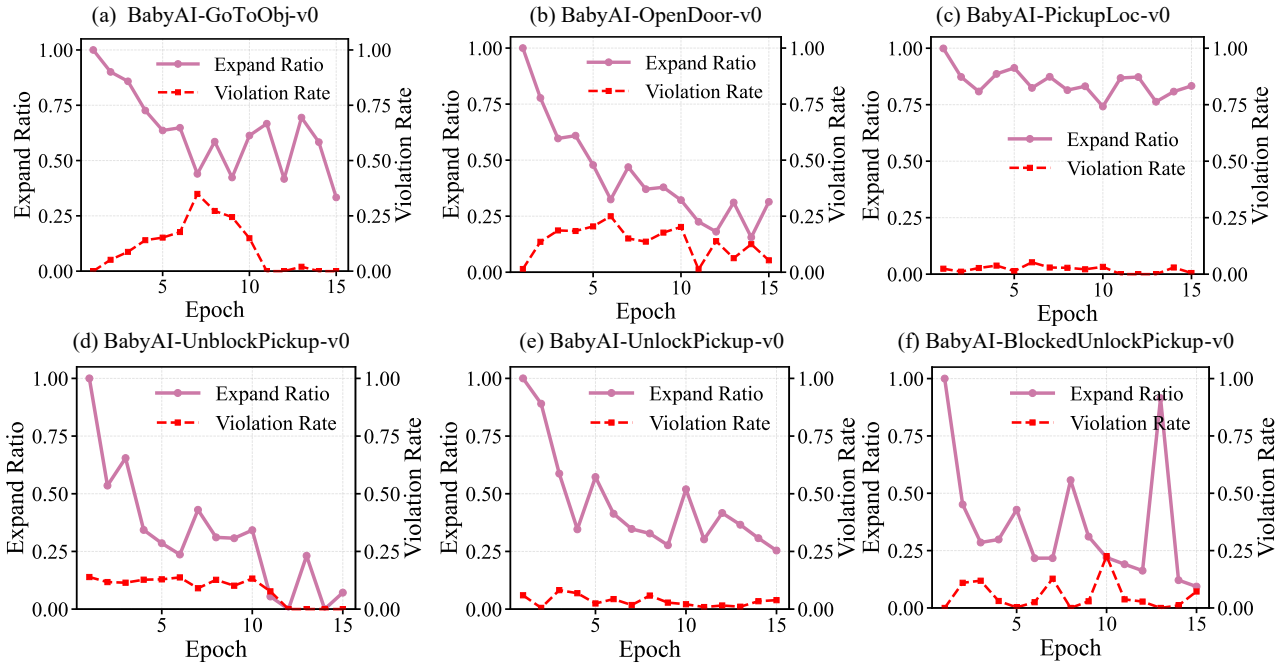


Figure 10. Expansion diagnostics across BabyAI training epochs. The expansion ratio measures the fraction of internal reasoning nodes expanded during lazy inference. The certificate violation rate measures the fraction of checked contracted internal nodes whose predicted certificate fails to cover the empirical energy gap between the contracted estimate and the fully expanded subtree energy.

Table 3. Temperature sensitivity of log-sum-exp energy aggregation. We evaluate $\tau \in \{0.25, 0.5, 1.0\}$ and use $\tau = 0.5$ as the default in the main experiments. Performance remains stable across this range, suggesting that the energy aggregation is not highly sensitive to the precise temperature choice.

Environment	$\tau = 0.25$	$\tau = 0.5$	$\tau = 1.0$
`GoToObj`	97%	98%	98%
`OpenDoor`	90%	93%	92%
`PickupLoc`	88%	91%	86%
`UnblockPickup`	82%	86%	84%

Table 4 summarizes the high-level root energies and semantic branches for each primitive action, and Figure 13 gives a compressed view of the corresponding action-conditioned recursive energy tree. The same construction principle is used across environments: each primitive action is associated with a reasoning tree whose root energy represents whether that action is appropriate under the current observation history and its imagined future.

The tree defines the factorization of the action energy. Internal nodes correspond to reusable semantic substructures, such as searching for a hidden target, improving navigation progress, clearing an obstacle, or executing a pickup. Leaf nodes correspond to grounded predicates that can be supervised during training. Figure 14 expands one such subtree for obstacle clearing, where temporary object manipulation is decomposed into initiating clearing, moving the obstacle, and safely dropping it. At inference time, the agent observes the current state, encodes the history, forms action-conditioned future representations, and predicts the energy of the relevant leaf and internal nodes. The recursive AND/OR aggregation then produces one energy value for each candidate action. The selected action is the one with the lowest resulting action energy, possibly after lazy expansion of uncertified subtrees.

Training and inference with the tree. The tree supplies a structured decomposition of action evaluation. During training, leaf predicates receive direct supervision from programmatic annotations derived from the environment state and subgoal annotations. These supervised leaves provide grounded local learning signals. Internal node energies and macro-energies are then learned through the recursive energy objective and wake–sleep compression, so that higher-level concepts such as navigation progress, obstacle clearing, and subgoal completion become reusable energy factors.

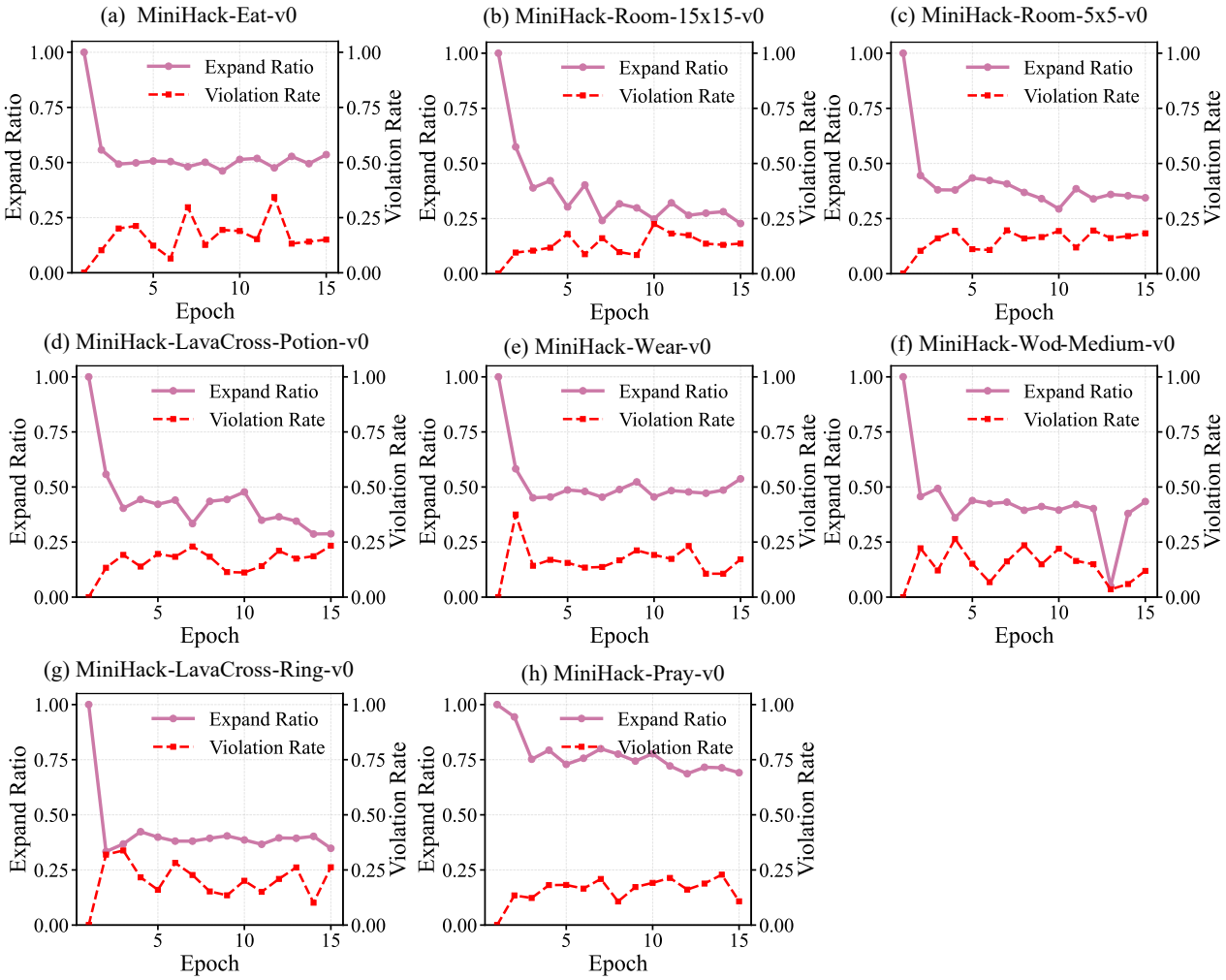


Figure 11. Expansion diagnostics across MiniHack training epochs. The expansion ratio measures the fraction of internal reasoning nodes expanded during lazy inference. The certificate violation rate measures the fraction of checked contracted internal nodes whose predicted certificate fails to cover the empirical energy gap between the contracted estimate and the fully expanded subtree energy.

During evaluation, the agent does not receive tree labels as input. It receives only the observation history and the current task context. The history encoder and action-conditioned future encoder produce representations for each candidate action. The model then predicts the leaf energies, internal energies, macro-energies, and contraction certificates from these representations. Lazy inference starts from contracted macro-evaluations and selectively expands subtrees when the current action winner is not yet certified. Thus, the same tree supports both fully expanded deliberative evaluation and contracted fast evaluation.

D. Implementation and Experimental Setup Details

D.1. Environment and Task Formulation

We evaluate the proposed low-level controller on two embodied symbolic-control benchmarks: BabyAI and MiniHack.

BabyAI (Chevalier-Boisvert et al., 2019) is a language-conditioned gridworld benchmark where agents solve navigation and object-interaction tasks through primitive actions such as turning, moving forward, picking up objects, dropping objects, and toggling doors. The BabyAI tasks used in our experiments emphasize object recognition, subgoal-conditioned navigation, local search, door interaction, unlocking, obstacle clearing, and target pickup.

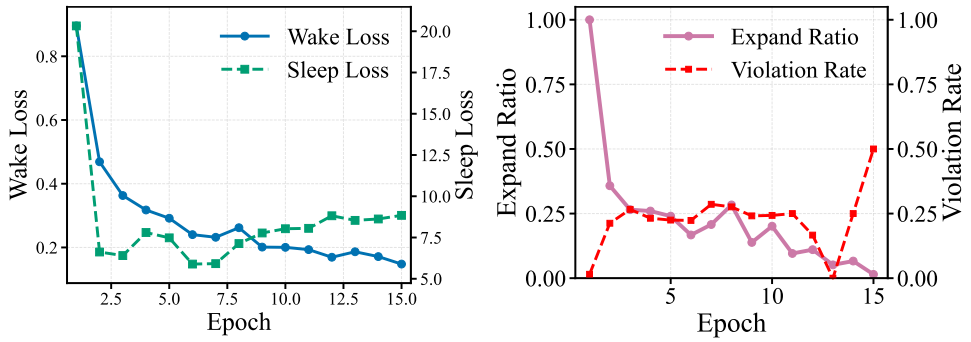


Figure 12. Example of over-contraction. Under an aggressive sleep-stage contraction setting, the expand ratio approaches zero by the final epoch, but the violation rate against fully expanded evaluation increases sharply.

Table 4. High-level structure of an example action-conditioned recursive energy tree.

Action	Root energy	High-level semantic branches
`left`	`should_turn_left_now`	obstacle clearing / target search / target navigation
`right`	`should_turn_right_now`	obstacle clearing / target search / target navigation
`forward`	`should_move_forward_now`	obstacle clearing / doorway progress / target search / target navigation
`pickup`	`should_pickup_now`	subgoal completion / obstacle-clearing initiation
`drop`	`should_drop_now`	safe drop / subgoal-feasibility improvement
`toggle`	`should_toggle_now`	door interaction / execution feasibility / immediate subgoal completion

MiniHack (Samvelyan et al., 2021) is a NetHack-based benchmark with richer symbolic observations, command-driven interactions, inventory-mediated behavior, and longer-horizon control. Our MiniHack tasks cover room navigation, eating, wearing equipment, praying, levitation-based lava crossing, and dungeon interactions. This benchmark provides a complementary testbed for checking whether the same recursive-energy controller can be instantiated beyond BabyAI-style gridworld tasks.

In both benchmarks, our focus is subgoal-conditioned low-level control rather than end-to-end high-level task decomposition. Given the current subgoal and the agent's partial observation, the learned controller selects the next primitive action. This setting tests whether dual-resolution reasoning can allocate computation adaptively: routine states may be handled by contracted habitual evaluation, while ambiguous states such as target search, doorway traversal, obstacle clearing, or command/inventory interactions may require selective expansion.

D.2. Hierarchical Setup with an LLM Planner

We instantiate the overall task as a hierarchical control problem. A high-level planner produces a sequence of symbolic subgoals, while the proposed learned model is responsible for low-level action selection under partial observation.

In the current implementation, the high-level planner is an LLM-based planner with a two-stage prompting procedure. The first stage performs *trajectory-to-subgoal distillation*. Given expert-bot trajectories, their missions, and compact symbolic summaries of the corresponding environments, the LLM analyzes how successful low-level behavior can be explained as high-level subgoal sequences. The admissible subgoal vocabulary is fixed in advance, and the LLM is asked to summarize expert behavior within this predefined space. This stage distills reusable high-level planning patterns, such as when a prerequisite object should be picked up, when a door should be crossed, when obstacle-clearing behavior is needed, and when the final target should be acquired.

The second stage performs *online instance-specific planning*. The planning prompt contains the distilled high-level planning summaries from expert trajectories, together with the current mission and the current fully observed symbolic environment state. During rollout, the LLM uses this information to output a concrete subgoal sequence for the current instance. If a previous high-level plan already exists, the prompt also includes the previous subgoal sequence, the current subgoal index, and the subgoals that have already been executed or completed. The LLM is then asked to decide whether the agent should continue the current subgoal or generate an updated plan from the current state.

Our role is to constrain the output interface by specifying the admissible subgoal vocabulary and the required JSON schema. For example, in Babyai environment, we restrict the high-level subgoal space to:

$$\{\texttt{CrossDoor}, \texttt{OpenDoor}, \texttt{MoveToTarget}, \texttt{Pickup}, \texttt{PutNext}\}.$$

The LLM planner is instructed to output strict JSON only. For trajectory distillation, the output is a structured analysis of expert behavior, including stage labels, temporal spans, purposes, and reusable planning summaries. For online planning, the output is a concrete subgoal plan, including the original mission, the chosen mission target, the active subgoal, and an ordered list of updated subgoals. Each subgoal contains a subgoal type, a concrete target object or location, a natural-language instruction, and metadata such as confidence and source.

During rollout, the LLM first produces an initial subgoal sequence. The plan is then periodically refreshed using the current global state, previous subgoal sequence, current subgoal index, and completed subgoals. The planner either keeps the current subgoal if it remains valid or replans when the subgoal is completed, invalid, or inconsistent with the realized execution state.

The two levels use different information scopes. The LLM planner has global state access for high-level task decomposition and replanning, whereas the learned low-level controller receives only the local partial observation, the current subgoal, and structured local features derived from observation history. Thus, the experimental stack consists of:

- a **high-level LLM planner** that distills expert-derived planning rules and updates symbolic subgoals during rollout;
- a **low-level learned controller** that selects primitive actions under partial observation conditioned on the current subgoal.

We treat the high-level planner as an external module and focus our evaluation on the proposed low-level recursive energy controller. The trajectory-distillation and online-replanning prompts are provided in Appendix F.

D.3. Dataset Construction and Supervision

Training data are constructed from successful demonstration trajectories. For BabyAI, we collect 400 successful expert-bot demonstrations per environment, corresponding to roughly 10^4 low-level expert steps, with small variation across environments. For MiniHack, we manually collect 40 successful trajectories per environment, corresponding to roughly 10^3 low-level steps. In both benchmarks, each low-level training example is aligned as a *pre-action* decision point: the input corresponds to the state before executing the demonstrated action, and the target is the action selected at that step.

The demonstrations provide low-level action supervision. We further annotate each demonstrated action with the high-level subgoal it serves. In BabyAI, these subgoal assignments are obtained retrospectively from the expert trajectory, environment state, and observed behavioral transitions. In MiniHack, the same interface is used, with subgoal annotations derived from manually collected trajectories and the command structure of each task.

We also construct leaf-level supervision for the reasoning trees. Leaf labels are derived from program state, the current annotated subgoal, local geometry, executability conditions, and action-conditioned imagined consequences. Thus, although BabyAI and MiniHack differ in how demonstrations are collected, the resulting training format is unified: each example contains a pre-action context, a demonstrated primitive action, a current subgoal, and grounded leaf-level predicate labels.

D.4. Training Objectives

The training pipeline is imitation-based. The model is trained to match demonstrated low-level actions while also learning grounded reasoning predicates and adaptive contraction behavior. The wake objective combines a lazy action loss computed from final partial action scores, an expanded auxiliary action loss computed from fully expanded scores, a binary cross-entropy leaf supervision loss over grounded predicate labels, and a small complexity penalty discouraging unnecessary expansion. The wake phase therefore teaches the model to choose the demonstrated action, align its predicate layer with grounded supervision, and avoid excessive explicit computation.

The sleep objective consolidates explicit System 2 reasoning into cheaper System 1 approximations. It replays traces collected during wake-time expansion and optimizes macro-energy distillation, which trains contracted macro-energies to approximate fully expanded subtree energies on visited nodes, together with certificate calibration, which trains certificates to cover the discrepancy between contracted and expanded evaluation without becoming overly conservative.

We use a three-stage training schedule. First, a warmup stage learns stable grounded predicate and action-scoring representations before lazy inference is used as the main decision pathway. Second, an expanded pretraining stage uses fully expanded action evaluation to stabilize the decision backbone and align the recursive tree with demonstrated primitive actions. Third, wake–sleep training jointly applies lazy action supervision, expanded auxiliary supervision, leaf supervision, complexity regularization, macro-energy distillation, and certificate calibration. In the current implementation, wake updates are frozen after a fixed number of epochs; wake inference continues to generate replay traces, while sleep continues refining macro-energies and certificates. This reflects the intended division of labor: once explicit reasoning behavior is sufficiently learned, later optimization focuses on compressing it into stable contracted evaluations.

D.5. Training Hyperparameters

Table 5 summarizes the default training and data settings. Table 6 reports the loss weights, and Table 7 reports model, lazy-inference, and sleep-stage hyperparameters. Most hyperparameters were kept fixed across environments; environment-specific changes, when used, are stated in the released configuration files.

Table 5. Default training and data hyperparameters.

Hyperparameter	Value
Batch size	32
Learning rate	1×10^{-3}
Weight decay	1×10^{-5}
Warmup epochs	5
Pretraining epochs	5
Joint training epochs	15
Freeze wake after epoch	15
Train ratio	0.8
Random seed	42
History length	6
Future horizon	1

Table 6. Loss weights and certificate calibration hyperparameters.

Hyperparameter	Value
λ_{action}	1.0
λ_{leaf}	0.5
$\lambda_{\text{pretrain_action}}$	1.0
$\lambda_{\text{pretrain_leaf}}$	0.5
λ_{macro}	1.0
λ_{cert}	1.0
λ_{comp}	0.01
$\lambda_{\text{wake_expanded_aux}}$	0.1
β_{cert}	0.01
λ_{under}	5.0
λ_{over}	0.25

E. Baseline Implementations

We implement all baselines under an input-aligned setting. In particular, the imitation-learning baselines receive the same subgoal-conditioned observation interface as our model, including the local image/history representation, structured local state features, and the current high-level subgoal. When available, baselines are also given one-step imagined futures generated by the same world-model transition module. Thus, the comparison is intended to isolate the effect of the recursive energy formulation and lazy contract–expand inference, rather than differences in privileged input information.

Flat-BC. The Flat-BC baseline is a supervised behavior-cloning policy without a hierarchical or recursive tree structure. It uses an image-history encoder to encode recent observations, previous actions, subgoal information, and local structured state features. The one-step imagined futures for all candidate primitive actions are flattened and projected into a compact future feature, which is concatenated with the history representation and passed to an MLP action head. The model is trained

Table 7. Model, lazy inference, and sleep-stage hyperparameters.

Hyperparameter	Value
State hidden dimension	256
Action hidden dimension	16
Leaf hidden dimension	32
History GRU hidden dimension	256
Future GRU hidden dimension	256
Scorer hidden dimension	256
OR temperature τ	0.5
Numerical ϵ	1×10^{-8}
Action certificate margin ϵ	1×10^{-6}
VoX ϵ	1×10^{-6}
Default contracted cost	0.05
Default expanded cost	0.50
Maximum lazy expansions	50
AND expansion threshold κ	4
Certificate initialization bias	60
Sleep steps per epoch	200
Replay buffer capacity	20000
Sleep batch size	32
Leaf class-balance weighting	Enabled
Leaf class-weight minimum count	1.0
Leaf class-weight maximum	80.0

to predict the expert primitive action using weighted cross-entropy, with higher weights assigned to important interaction stages such as door manipulation and obstacle clearing. We train for 20 epochs using AdamW, with learning rate 10^{-3} and batch size 32, and select the checkpoint with the best validation action accuracy.

H-BC. The H-BC baseline is a supervised behavior-cloning model that keeps the same high-level subgoal conditioning and action-candidate future interface, but does not use recursive energy composition. It first encodes the observation/action history, current and previous subgoals, and local structured state into a 256-dimensional history representation. For each candidate primitive action, a future encoder maps the corresponding one-step imagined future into an action-specific future embedding. The concatenation of the history representation and all action future embeddings is passed to an MLP action head trained with standard cross-entropy against the expert action. We train for 20 epochs using AdamW with learning rate 10^{-3} and batch size 64, and select the checkpoint with the best validation action accuracy.

RL. The RL baseline is a subgoal-conditioned actor–critic policy trained with PPO. It uses the same high-level subgoal interface, recent observation/action history, local structured features, and one-step imagined futures as the imitation baselines. The actor predicts primitive-action probabilities, while the critic estimates state values. Training uses environment rewards plus a subgoal completion bonus. Unless otherwise specified, the PPO implementation uses a rollout buffer of 2048 steps, four PPO epochs per update, clipping coefficient 0.2, discount $\gamma = 0.99$, GAE parameter 0.95, entropy coefficient 0.01, and learning rate 3×10^{-4}. In the reported comparisons, RL baselines use a 1M-step environment interaction budget.

Expert-initialized PPO for difficult stages. For the difficult `BlockedUnlockPickup` setting, we additionally consider an expert-initialized subgoal-conditioned PPO baseline. This baseline first performs behavior-cloning warm-up on 1000 expert demonstration episodes, then continues PPO fine-tuning online. The architecture remains actor–critic and uses the same subgoal-conditioned history, local-structure, and imagined-future interface as above, with additional task-specific context features for key phases such as crossing, unlocking, and pickup. The PPO fine-tuning stage uses environment reward plus a subgoal completion bonus, with learning rate 5×10^{-5}, buffer size 8192, four PPO epochs per update, clipping coefficient 0.2, discount $\gamma = 0.99$, and entropy coefficient 0.015. This baseline is intended to test whether expert initialization plus online RL can improve difficult key-action sequences.

F. Prompt for LLM planner

Box 1: Prompt for LLM-Based High-Level Stage Distillation

You are analyzing expert trajectories in a BabyAI-like grid world.
Your task is to infer reusable high-level stages from expert trajectories. The stage space is fixed in advance. You must assign each inferred stage to one of the following allowed labels:

$$\{\texttt{CrossDoor}, \texttt{OpenDoor}, \texttt{MoveToTarget}, \texttt{Pickup}, \texttt{PutNext}\}.$$

You should infer stage structure from:

- the mission,
- the initial environment/program state,
- and the low-level expert action trajectory.

Guidelines:

- Segment the trajectory into a small number of consecutive stages.
- Each stage should describe an intention-level behavior, not primitive movements.
- Use only the allowed stage labels above.
- Do not invent new stage names.
- Keep label usage consistent across examples.
- Distinguish temporary enabling stages from final task-completion stages.
- If several actions serve one larger purpose, merge them into one stage.
- Use the initial state to explain why the trajectory takes this form.
- Output only valid JSON.

Stage label meanings:

- `CrossDoor`: get through a target door into the next room. This may include approaching, unlocking/opening, and crossing the doorway.
- `OpenDoor`: open a door when opening itself is the semantic goal.
- `MoveToTarget`: move to a target object or location without picking it up or using it.
- `Pickup`: pick up a target object or a necessary prerequisite object, such as a key.
- `PutNext`: place one object next to another target object.

Return format:

```
{
  "stages": [
    {
      "stage_id": 0,
      "label": "CrossDoor | OpenDoor | MoveToTarget | Pickup | PutNext",
      "purpose": "...",
      "start_step": 0,
      "end_step": 0,
      "temporary_or_final": "temporary | final"
    }
  ],
  "overall_strategy": "...",
  "suggested_reusable_labels": [
    {
      "label": "CrossDoor | OpenDoor | MoveToTarget | Pickup | PutNext",
      "definition": "..."
    }
  ]
}
```

Few-shot examples:

```
[EXAMPLE i]

Mission:
<MISSION>

Initial state:
<REDUCED_INITIAL_PROGRAM_STATE>

Trajectory:
<LOW_LEVEL_ACTION_SEQUENCE>

Summary:
<GOLD_OR_PREVIOUSLY_DISTILLED_SUMMARY>
```

Now analyze the new case:

```
Mission:
<TEST_MISSION>

Initial state:
<TEST_REDUCED_INITIAL_PROGRAM_STATE>

Trajectory:
<TEST_LOW_LEVEL_ACTION_SEQUENCE>
```

Example: Standardized LLM-Distilled Stage Output


```
{
  "stages": [
    {
      "stage_id": 0,
      "label": "Pickup",
      "purpose": "Acquire the grey key required to unlock the grey door.",
      "start_step": 0,
      "end_step": 4,
      "temporary_or_final": "temporary"
    },
    {
      "stage_id": 1,
      "label": "CrossDoor",
      "purpose": "Use the carried grey key to unlock/open the grey door and pass through it.",
      "start_step": 5,
      "end_step": 15,
      "temporary_or_final": "temporary"
    },
    {
      "stage_id": 2,
      "label": "Pickup",
      "purpose": "Approach and pick up the target box to complete the mission.",
      "start_step": 16,
      "end_step": 23,
      "temporary_or_final": "final"
    }
  ],
  "overall_strategy": "The expert first picks up the key needed for the locked door, then crosses the
      door to access the target area, and finally picks up the mission target.",
  "suggested_reusable_labels": [
    {
      "label": "Pickup",
      "definition": "Acquire a target or prerequisite object, such as a key or the final mission
     object.",
      "typical_trigger": "An object must be carried to enable later progress or complete the mission
     .",
      "typical_completion": "The agent is carrying the specified object."
    },
    {
      "label": "CrossDoor",
      "definition": "Traverse a specific door into the next region, including prerequisite unlocking
     or opening when necessary.",
      "typical_trigger": "The target is behind a door or passage that must be crossed.",
      "typical_completion": "The agent has passed through the door into the target-side region."
    }
  ]
}
```

Example: Standardized LLM-Distilled Stage Output


```
{
  "stages": [
    {
      "stage_id": 0,
      "label": "Pickup",
      "purpose": "Acquire the grey key required to unlock the grey door.",
      "start_step": 0,
      "end_step": 4,
      "temporary_or_final": "temporary"
    },
    {
      "stage_id": 1,
      "label": "CrossDoor",
      "purpose": "Use the carried grey key to unlock/open the grey door and pass through it.",
      "start_step": 5,
      "end_step": 15,
      "temporary_or_final": "temporary"
    },
    {
      "stage_id": 2,
      "label": "Pickup",
      "purpose": "Approach and pick up the target box to complete the mission.",
      "start_step": 16,
      "end_step": 23,
      "temporary_or_final": "final"
    }
  ],
  "overall_strategy": "The expert first picks up the key needed for the locked door, then crosses the
      door to access the target area, and finally picks up the mission target.",
  "suggested_reusable_labels": [
    {
      "label": "Pickup",
      "definition": "Acquire a target or prerequisite object, such as a key or the final mission
     object.",
      "typical_trigger": "An object must be carried to enable later progress or complete the mission
     .",
      "typical_completion": "The agent is carrying the specified object."
    },
    {
      "label": "CrossDoor",
      "definition": "Traverse a specific door into the next region, including prerequisite unlocking
     or opening when necessary.",
      "typical_trigger": "The target is behind a door or passage that must be crossed.",
      "typical_completion": "The agent has passed through the door into the target-side region."
    }
  ]
}
```


Box 2: Prompt for Online LLM Subgoal Planning and Replanning

Role. You are an online high-level planner for BabyAI-style grid environments.

Goal. Given the current mission, the current fully observed symbolic environment state, the planning logic distilled from expert-bot trajectories, and the previous high-level plan, decide whether the agent should continue the current subgoal or replan. If replanning is needed, generate an updated concrete subgoal sequence for the low-level controller.

Allowed subgoal types. Use only:

$$\{\texttt{CrossDoor}, \texttt{OpenDoor}, \texttt{MoveToTarget}, \texttt{Pickup}, \texttt{PutNext}\}.$$

Planning instructions.

- Do not output primitive actions.
- Do not invent new subgoal types.
- Use the distilled planning rules as the main source of high-level planning logic.
- Use exact object attributes and coordinates from the current environment state.

- Consider the previous subgoal sequence, the current subgoal index, and the subgoals already completed.
- If the current subgoal is still valid and useful, keep it and set `decision` to `continue`.
- If the current subgoal is completed, invalid, blocked by a newly relevant prerequisite, or inconsistent with the current state, set `decision` to `replan`.
- When replanning, preserve useful unfinished subgoals from the previous plan when possible.
- Add prerequisite subgoals when required by the current map state.
- If a locked door blocks progress, insert the required `Pickup` subgoal for the key before door traversal.
- Use `CrossDoor` when the agent must enter another room through a door.
- Use `OpenDoor` only when opening a door is the direct semantic objective.
- Use `Pickup` for the final target object when the mission requires acquiring it.
- Use `PutNext` when the mission requires placing one object next to another.
- Output strict JSON only.

Required output schema.

```
{
  "original_mission": "...",
  "decision": "continue | replan",
  "decision_reason": "...",
  "current_subgoal_index": 0,
  "active_subgoal": {
    "subgoal_id": 0,
    "subgoal_type": "CrossDoor | OpenDoor | MoveToTarget | Pickup | PutNext",
    "target": {
      "x": 0,
      "y": 0,
      "type": "...",
      "color": "..."
    },
    "instruction": "...",
    "confidence": "high | medium | low",
    "source": "llm_online_planner"
  },
  "mission_target": {
    "type": "...",
    "color": "...",
    "candidates": [
      {"x": 0, "y": 0, "type": "...", "color": "..."}
    ],
    "chosen": {"x": 0, "y": 0, "type": "...", "color": "..."},
    "choice_source": "llm_from_fully_observed_map"
  },
  "updated_subgoals": [
    {
      "subgoal_id": 0,
      "subgoal_type": "CrossDoor | OpenDoor | MoveToTarget | Pickup | PutNext",
      "target": {
        "x": 0,
        "y": 0,
        "type": "...",
        "color": "..."
      },
      "instruction": "...",
      "status": "completed | active | pending",
      "confidence": "high | medium | low",
      "source": "llm_online_planner",
      "event_t": -1,
      "num_steps_est": null
    }
  ],
  "completed_subgoals": [
    {
      "subgoal_id": 0,
      "subgoal_type": "..."
    }
  ],
  "temporary_pickdrop_spans": []
}
```

Planning context.

```
Mission:
<CURRENT_MISSION>

Current symbolic environment state:
<CURRENT_PROGRAM_STATE_SUMMARY>

Expert trajectory distillation:
<LLM_SUMMARIES_FROM_EXPERT_TRAJECTORIES>

Distilled planning rules:
<PLANNING_RULES_FROM_TRAJECTORY_DISTILLATION>

Previous high-level plan:
<PREVIOUS_SUBGOAL_SEQUENCE_OR_EMPTY_IF_INITIAL>

Current subgoal index:
<CURRENT_SUBGOAL_INDEX_OR_NULL_IF_INITIAL>

Completed subgoals:
<COMPLETED_SUBGOALS_OR_EMPTY_IF_INITIAL>

Previously executed subgoals:
<EXECUTED_SUBGOAL_HISTORY_OR_EMPTY_IF_INITIAL>
```

G. Autonomous LLM Tree Induction and Repair Pipeline

This appendix reports a BabyAI instantiation of a more general LLM-assisted tree-induction pipeline. The general pipeline only assumes demonstrations, a task interface specifying observable state fields, an action set, and an executable verifier for action-conditioned trees. The concrete predicates, observation fields, subgoal descriptors, and primitive actions below are specific to the BabyAI case study.

We use a constrained multi-agent LLM pipeline to induce and repair the action-conditioned energy tree from local demonstrations. The LLM components propose symbolic predicates, label observable predicate values, triage verifier disagreements, and suggest tree edits. The executable verifier remains symbolic: it evaluates the tree on predicate labels and determines the minimum-energy action set. This separation is important because the induced tree is intended to provide a locally reasonable candidate action set rather than imitate every demonstrated action with a single deterministic preference.

Bootstrap induction. The pipeline first selects two demonstration subsets. Tree-building examples retain expert actions and are used for predicate and tree induction, whereas covering examples are split into a labeler view that hides the current expert action and a verifier view that keeps it for evaluation. A predicate-library agent then reads the task semantics and a small set of local-observation demonstrations to produce a reusable library of observable Boolean predicates. The library includes primitive state facts, subgoal facts, visibility and memory facts, inventory facts, local affordances, and context gates for search, navigation, obstacle clearing, immediate interaction, and door traversal. A tree-builder agent receives this library and demonstrations and constructs an initial action-conditioned AND/OR tree over the environment's primitive action set. In the BabyAI instantiation, this action set is $\{\texttt{left}, \texttt{right}, \texttt{forward}, \texttt{pickup}, \texttt{drop}, \texttt{toggle}\}$. Each leaf must reference a predicate from the library. The generated tree is normalized and schema-repaired before it enters the multi-round repair loop.

Label-and-verify loop. At each repair round, the predicate labeler annotates the predicates used by the current tree for every step in the covering episodes. The labeler uses only the current local observation, current subgoal, observable inventory, and bounded local history; it is not given the current expert action. The recursive energy verifier maps each predicate label to a leaf energy,

$$E(\ell) = \begin{cases} 0, & \text{if the predicate at leaf } \ell \text{ is labeled true,} \\ 100, & \text{otherwise,} \end{cases}$$

aggregates AND nodes by summation and OR nodes by minimum,

$$E_{\mathrm{AND}} = \sum_i E_i, \qquad E_{\mathrm{OR}} = \min_i E_i,$$

and returns the minimum-energy action set

$$\mathcal{A}_{\min}(o_t) = \left\{ a : E_a(o_t) = \min_{a'} E_{a'}(o_t) \right\}.$$

The verifier stores both strict action agreement and relaxed tie agreement. The latter records whether the expert action is contained in $\mathcal{A}_{\min}(o_t)$ and is the more relevant signal for a candidate-set tree.

Disagreement triage and repair. Verifier disagreements are not repaired blindly. A judgment agent inspects strict mismatches using the local observation context, expert action, and minimum-energy action set. It marks acceptable ties and potentially suboptimal demonstration actions as weak repair signals, while retaining cases in which a reasonable expert action is missing from the candidate set or the candidate set contains locally unreasonable actions. A predicate-gap agent then checks whether these actionable cases require new observable predicates, predicate definition repair, or only tree-structure repair. Finally, a tree-repair agent proposes local patch operations, such as adding a context-specific AND branch or adding a guard predicate to an existing branch. Candidate patches are applied greedily and verified on existing predicate labels before being retained. The current acceptance score prioritizes relaxed candidate-set coverage, gives only a small bonus to strict expert agreement, and penalizes degenerate all-action ties.

G.1. Repair Outcome

We report the repair dynamics and downstream policy results of the BabyAI `UnblockPickup` case study. Figure 15 shows one representative autonomous repair run, illustrating how verifier metrics evolve over successive tree edits. Table 8 summarizes aggregate downstream performance and diagnostic statistics over repeated LLM-induced trees, together with the human-checked tree used in the controlled main experiments.

Verifier metrics. For each labeled step t, the symbolic verifier computes one root energy $E_a(o_t)$ for every primitive action $a \in \mathcal{A}$ and returns the minimum-energy candidate set

$$\mathcal{A}_{\min}(o_t) = \{a \in \mathcal{A} : E_a(o_t) = \min_{a' \in \mathcal{A}} E_{a'}(o_t)\}.$$

Let $a_t^\star$ denote the demonstrated expert action and let $\hat{a}_t$ denote the verifier's deterministic predicted action after tie breaking. We report the following diagnostics over N labeled steps:

$$\text{Strict Acc.} = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}[\hat{a}_t = a_t^\star],$$

which measures exact agreement with the demonstrated action, and

$$\text{Relaxed Tie Acc.} = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}[a_t^\star \in \mathcal{A}_{\min}(o_t)],$$

which measures whether the expert action is included in the verifier's minimum-energy candidate set. The relaxed metric is appropriate because the tree is designed to produce a locally reasonable candidate set, not necessarily to imitate one deterministic demonstration trace when multiple actions are equally reasonable.

To check that relaxed agreement is not obtained by overly broad ties, we also track candidate-set diagnostics. The average tie width is

$$\text{AvgTieWidth} = \frac{1}{N} \sum_{t=1}^{N} |\mathcal{A}_{\min}(o_t)|.$$

The wide-tie rate measures the fraction of steps whose minimum-energy candidate set is broader than a normal small tie,

$$\text{WideTieRate} = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}[|\mathcal{A}_{\min}(o_t)| > \kappa],$$

where κ is a fixed threshold used only as a degeneracy diagnostic. The all-action tie rate is the stricter failure mode

$$\text{AllActionTieRate} = \frac{1}{N}\sum_{t=1}^{N} \mathbf{1}[|\mathcal{A}_{\min}(o_t)| = |\mathcal{A}|],$$

which detects cases where the verifier assigns every action to the minimum-energy set. Finally, internal nodes are computed as the number of total tree nodes minus the number of leaf predicate nodes, and are used as a simple measure of induced tree complexity.

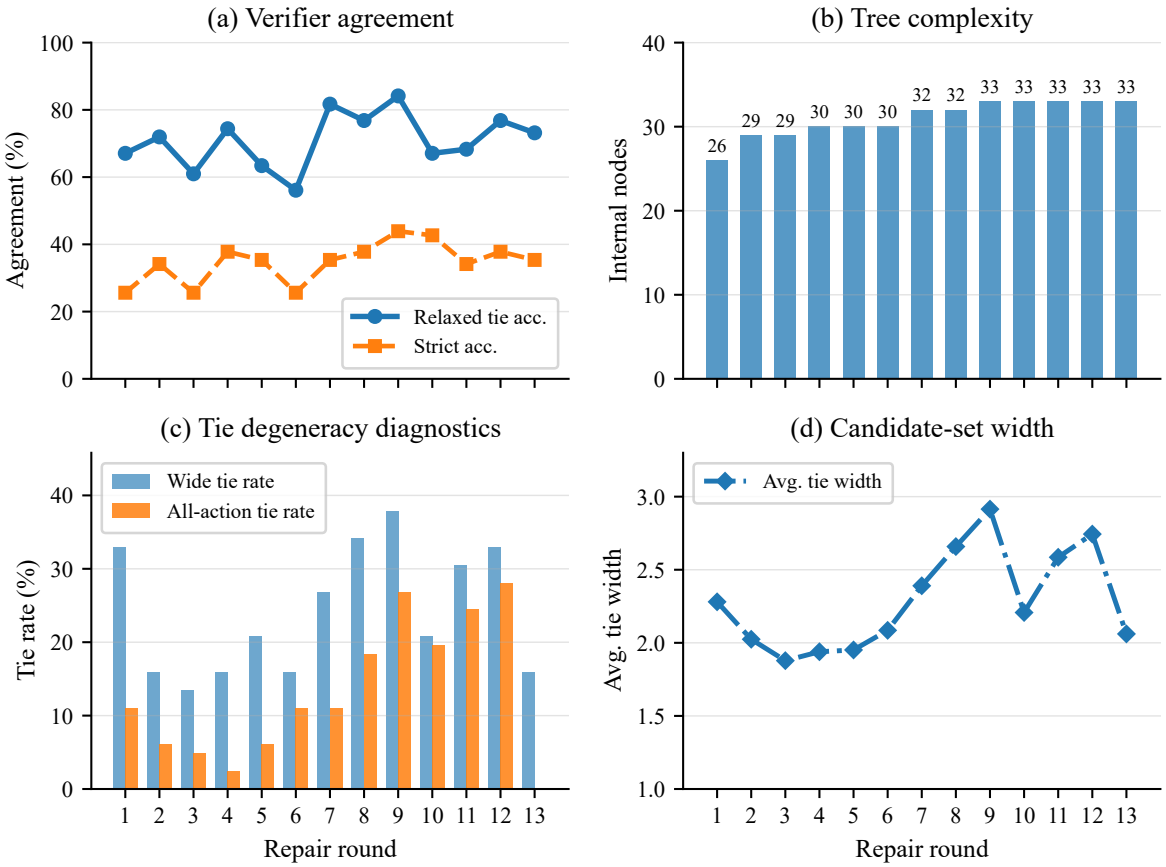


Figure 15. Representative repair dynamics of the autonomous LLM tree induction and repair pipeline on the BabyAI `UnblockPickup` case study. The top-left panel shows strict and relaxed verifier agreement, the top-right panel shows the growth of internal reasoning nodes, and the bottom panels monitor candidate-set degeneracy through tie rates and average tie width. These diagnostics show that the relaxed-tie improvement is tracked together with tree complexity and candidate-set width, rather than being reported as an isolated agreement metric.

Table 8. Outcome of the autonomous LLM tree induction and repair pipeline on the BabyAI `UnblockPickup` case study. We report mean±std over repeated LLM-induced trees without listing individual runs. The human-checked baseline uses a fixed manually checked tree and is evaluated over repeated policy runs. Policy success is measured on `UnblockPickup`; tree nodes denote the number of nodes in the resulting action-conditioned tree.

Metric	LLM-induced trees	Human-checked tree
Policy success on `UnblockPickup`	82.0±4.4%	83.3±6.4%
Tree nodes	31.3±2.1	35
Predicate agreement	93.5±3.8%	–
Average tie width	2.83±0.40	–
Wide tie rate	50.8±32.5%	–
All-action tie rate	12.6±13.5%	–

Across repeated LLM-induced trees, the autonomous pipeline achieves 82.0±4.4% policy success with compact trees of 31.3±2.1 nodes, which is comparable to the 83.3±6.4% success of the human-checked tree with 35 nodes. The induced trees also obtain high predicate agreement (93.5±3.8%). Although tie diagnostics vary across induced trees, the all-action tie rate remains explicitly monitored, reducing the risk that candidate-set agreement is obtained by trivially assigning all actions to the minimum-energy set. These results suggest that human checking improves stability and provides a controlled setup for the main experiments, but the framework does not fundamentally require a fully hand-engineered tree.

G.2. Prompts Used by the LLM Agents

We report the fixed instruction prompts used in the implementation with minor formatting edits only. Dynamic JSON payloads such as demonstrations, current tree JSON, predicate-library JSON, verifier counterexamples, and label jobs are not reproduced in full. We state the omitted payload fields below each prompt excerpt.

Box A1: Prompt for LLM Predicate-Library Induction

Prompt excerpt.

```
You are an expert predicate-library designer for symbolic action-tree induction.
Your task:
Given demonstrations from a BabyAI-style gridworld environment, define a
reusable library of observable boolean predicates.
Important:
- Predicates should describe environment facts, not action preferences.
- Predicates should be reusable across action trees.
- Predicates must be labelable from:
1. partial˙obs,
2. current˙subgoal,
3. agent˙inventory,
4. short local history.
- agent˙inventory is observable agent state. It tells whether the agent is
carrying an object and, if so, the carried object's type and color.
- Predicates must NOT depend on future observations.
- Predicates must NOT mention expert˙action, predicted˙action, timestep, seed,
episode˙id, or demonstration identity in their true˙when / false˙when.
- Predicates should be stable, simple, and compositional.
- Prefer primitive factual predicates over overly specific policy predicates.
Good predicate examples:
- p˙subgoal˙is˙pickup
- p˙target˙visible˙left
- p˙front˙closed˙door
- p˙agent˙carrying˙non˙target˙object
- p˙safe˙drop˙front
Bad predicate examples:
- p˙should˙turn˙left˙now
- p˙expert˙action˙is˙left
- p˙step˙12˙case
```

- p˙predicted˙action˙wrong
- p˙future˙target˙will˙appear
Return valid JSON only.

Dynamic payload omitted from the excerpt. The user message supplies task semantics, a compact observation legend, predicate-family examples and design rules, a JSON output schema, and compact tree-building demonstrations.

Box A2: Prompt for Initial Energy-Tree Induction

Prompt excerpt.

You are an expert in program synthesis, symbolic planning, and interpretable
low-level control.
Your task is to induce an action-conditioned AND/OR reasoning tree for a BabyAI
low-level controller.
Critical predicate-library constraint:
- You will be given a predicate˙library.
- You MUST use only predicate names that appear in predicate˙library.predicates.
- Do NOT invent new predicates.
- Do NOT rename predicates.
- Do NOT redefine predicate semantics.
- If you think an important predicate is missing, list it under
requested˙missing˙predicates, but do NOT use it in any LEAF node.
Action-tree constraints:
- The tree must be action-conditioned: each primitive action has its own root
reasoning tree.
- Allowed actions are exactly: left, right, forward, pickup, drop, toggle.
- Internal nodes must be AND or OR.
- Leaf nodes must have type LEAF and reference one existing predicate from
predicate˙library.predicates.
- Use only information available in local 7x7 observation text, current
subgoal, and short local history.
- Do NOT use full-map information, global coordinates, hidden object locations,
shortest paths, or privileged simulator state.
- Do NOT use episode˙id, seed, timestep, expert˙action, or action˙sequence
inside predicate definitions or node logic.
- Expert actions are shown only to help infer tree structure.
Energy semantics:
- A true leaf has energy 0.
- A false leaf has energy 100.
- AND node energy is the sum of child energies.
- OR node energy is the minimum of child energies.
- The selected action is the action whose root has the lowest energy.
Your output must be a single valid JSON object only.

Dynamic payload omitted from the excerpt. The initial-tree user payload provides task semantics, energy semantics, the compact predicate library, the required tree JSON schema, and demonstrations. Its core requirements instruct the model to prefer context-specific AND branches under OR action roots and to separate search, navigation, obstacle clearing, immediate interaction, and door traversal.

Box A3: Prompt for LLM Predicate Labeling

Prompt excerpt.

You are a careful predicate-labeling assistant for an interpretable BabyAI
low-level controller.
Task:
Given one full local-observation episode, label boolean predicates for every
step.
Allowed information:
- current local 7x7 observation text at the step being labeled
- current subgoal at the step being labeled
- agent˙inventory at the step being labeled, including whether the agent is
carrying an object and its visible type/color if available

- agent˙state at the step being labeled, if provided; it is a compact local/self state without global coordinates
- action˙conditioned˙future˙by˙t[str(t)], if provided; it is a one-step counterfactual local-effect summary for all actions, not the observed future trajectory
- previous local trajectory context up to HISTORY˙MAX˙LEN
- previous executed actions before the current step, if provided in history˙by˙t

Forbidden information:
- current expert action
- future expert actions
- observed future episode steps after the current step
- full grid
- global coordinates
- hidden object locations
- full program˙state
- raw hand-crafted state˙struct features other than already extracted local agent inventory, agent state, or local effects

Important:
- The full episode is provided only for efficient offline annotation.
- When labeling step t, conceptually use only the current step and bounded previous context.
- Do not infer expert action.
- If local evidence is insufficient or ambiguous, label 0.
- Output valid JSON only.

Dynamic payload omitted from the excerpt. Each label job supplies the requested predicate names and definitions, the episode-local fields allowed above, bounded history for each step, and the JSON schema for complete $0/1$ labels by step.

Box A4: Prompt for Disagreement Judgment

Prompt excerpt.

You are an expert disagreement triage assistant for BabyAI local-control verification.
You do NOT repair the tree.
You do NOT repair predicates.
You do NOT need to read the full action tree.
Your task:
Given local observation context, current subgoal/mission, expert action, action energies, and tied˙min˙actions, judge whether the disagreement should be used as a repair signal.
Important cases:
1. If expert˙action is inside tied˙min˙actions:
- This is already relaxed-tie correct.
- Usually it is acceptable, even if the energy is not zero.
- Only request repair if tied˙min˙actions contains clearly unreasonable actions.
2. If expert˙action is not inside tied˙min˙actions:
- First judge whether expert˙action is reasonable under the current local observation and subgoal.
- Also judge whether the actions in tied˙min˙actions are reasonable.
- If expert is unreasonable and min-energy actions are reasonable, mark as demo˙noise˙or˙expert˙suboptimal and do not repair.
- If expert is reasonable and min-energy actions are unreasonable or incomplete, mark as actionable.
3. If the agent already sees the relevant target/door/front object clearly, search-like turns may be unreasonable unless they align with the target or avoid blockage.
4. If the front cell is blocked/wall/closed door/object, forward may be unreasonable unless opening/crossing is possible and appropriate.
5. If pickup/drop/toggle is involved, judge whether the front object/door and subgoal make that interaction reasonable.
You may use traces only as energy evidence. Do not propose exact tree edits.

Return valid JSON only.

Dynamic payload omitted from the excerpt. The judger receives compact verifier mismatches grouped by episode, including local context, subgoal and mission, expert action, predicted action, action energies, tied minimum-energy actions, and compact verifier traces.

Box A5: Prompt for Predicate-Gap Diagnosis

Prompt excerpt.

You are an expert diagnostic assistant for predicate-library based symbolic control.
Your task:
Given a fixed predicate library, a current action tree, and verifier counterexamples, decide whether failures are caused by missing/unclear predicates or by tree structure.
Failure types:
1. missing˙predicate
2. bad˙predicate˙definition
3. bad˙label
4. bad˙tree˙structure
5. tie˙or˙energy˙issue
Context diagnosis:
- For each actionable failure, identify the intended decision context when possible: search, navigation, obstacle˙clearing, immediate˙interaction, or door˙traversal.
- If a predicted action is cheap because a branch fires in the wrong context, classify as bad˙tree˙structure or tie˙or˙energy˙issue.
- If the expert action lacks support in the correct context, classify as bad˙tree˙structure.
- If the library lacks any observable way to distinguish the needed context, classify as missing˙predicate.
- Do not request a new predicate merely because the current tree failed to use an existing useful predicate.
Rules:
- Do not invent hidden-state predicates.
- Missing predicates must be observable from local observation, current subgoal, agent˙inventory, or short local history.
- If existing predicates are sufficient, choose tree˙repair˙only.
Return valid JSON only.

Dynamic payload omitted from the excerpt. Predicate-gap diagnosis receives the compact predicate library, current tree, coverage summary, actionable counterexample batches, and judgment summary.

Box A6: Prompt for Patch-Based Tree Repair

Prompt excerpt.

You are an expert program-synthesis assistant.
Your task:
Given the current action-conditioned AND/OR tree, fixed predicate˙library, predicate-gap diagnosis, disagreement judgment, and batched repair notes, produce PATCH OPERATIONS, not a full repaired tree.
Critical constraints:
- Output JSON patch operations only.
- Use only predicates from predicate˙library.predicates.
- Do NOT invent new predicates, rename predicates, or redefine predicate semantics.
- Prefer small local edits.
- Preserve the existing tree unless a patch explicitly changes one branch.
Allowed patch ops:
1. add˙and˙branch˙to˙action˙root
2. add˙guard˙predicate˙to˙branch
3. replace˙predicate˙leaf

4. remove˙child˙from˙branch
Energy semantics:
- true leaf energy = 0
- false leaf energy = 100
- AND node energy = sum child energies
- OR node energy = min child energies
- selected action = action with minimum root energy
Repair strategy:
- Use disagreement˙judgment to avoid repairing demo noise or acceptable ties.
- Optimize the minimum-energy candidate set, not just strict imitation of one expert trace.
- A tie is acceptable when all tied actions are locally reasonable and the reasonable expert action is included.
- Do not remove a plausible alternative action only to match the expert's exact trajectory.
- Do repair ties that include clearly unreasonable interaction actions or actions from the wrong context.
Return valid JSON only.

Dynamic payload omitted from the excerpt. Patch repair receives the current tree, fixed predicate library, diagnosis, judgment, validation summary, coverage summary, and compact repair notes.

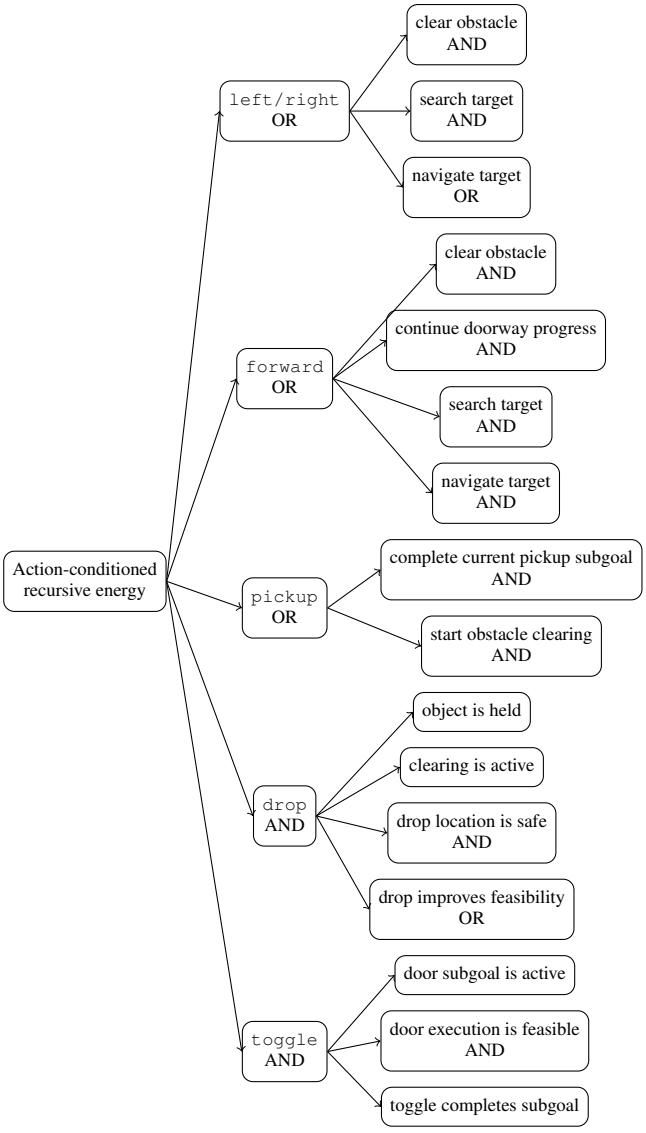


Figure 13. Compressed view of an example action-conditioned recursive energy tree. Each primitive action has a root energy, and the root is recursively decomposed into semantic branches and grounded predicates. During inference, the agent predicts node energies from the current observation history and action-conditioned future representation, then aggregates them recursively to score candidate actions.



Figure 14. Expanded view of the obstacle-clearing factors in the example tree. The subtree decomposes temporary object manipulation into initiating clearing, moving the obstacle, and safely dropping it. These factors allow the learned energy to distinguish ordinary navigation from temporary manipulations that support a later pickup or doorway-crossing subgoal.