DUAL-RESOLUTION RECURSIVE ENERGY FOR SELECTIVE DELIBERATION IN SEQUENTIAL DECISION MAKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Fast action selection and slow deliberation are often modeled as separate computational modes. We ask whether they can instead be realized as two evaluation resolutions of a single structured decision model. We introduce a dual-resolution recursive energy model in which contracted macro-energies and explicitly composed AND/OR subtrees evaluate the same action-support structure on a common energy scale. Local expansion controls the extent of explicit deliberation within a decision. Wake–sleep training distills expanded subtree energies into reusable macro-energies and learns estimates of their contraction errors. At inference, these estimates guide which frontier nodes to expand and when the action-energy margin permits stopping. The fully expanded winner is preserved when the predicted errors cover the relevant contraction gaps, or when any remaining undercoverage fits within the action margin. Across BabyAI, MiniHack, and Overcooked-AI, the model reduces explicit recursive expansion by 50.88–77.74% while retaining near-Full macro success, matching Full on Overcooked-AI. It achieves higher macro success than the matched-budget controls in the benchmark-level summaries. On identical held-out decision states, Ours reduces task-averaged median action-selection latency by 26.6–57.5% relative to Full while retaining at least 99.2% task-averaged action agreement.

1 INTRODUCTION

An action-conditioned decision model may evaluate many supporting conditions even when only a few can affect the selected action. The central problem is to retain a detailed, compositional action energy while deciding which local factors need explicit evaluation at each step. The same primitive action can call for different resolution across contexts: moving through an open corridor may be resolved from a learned summary, whereas approaching a locked door may require evaluating access, available objects, and alternative routes. The contrast is reminiscent of fast and deliberative processing in dual-process accounts (Kahneman, 2011; Evans & Stanovich, 2013). Following this terminology, we use *System 1* to denote fast, compressed evaluation that relies on learned summaries, and *System 2* to denote slower deliberative evaluation that exposes more of the underlying decision structure. We use this distinction as an operational analogy rather than a cognitive claim.

A quick evaluation should reuse what the agent has learned about familiar parts of a decision, while deliberation should expose the details that remain unresolved. Both can be useful at once: moving toward an object may involve familiar navigation but an uncertain access condition. Such deliberation can be viewed as selectively exposing the factors that support a decision: familiar situations allow these factors to be summarized, while ambiguous situations require their explicit evaluation. This calls for selective inspection of the unresolved component while the familiar part remains compressed. The relevant unit of adaptive effort is therefore a local decision subproblem. In our model, leaf predicates encode decision-relevant conditions in the current context, such as whether an obstacle blocks progress or an object is held. AND/OR nodes compose required conditions and alternative sources of support for a candidate action. Expansion explicitly evaluates more of this structure; its extent serves as an operational proxy for the agent's deliberation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Computational dual-process models can use distinct controllers connected by arbitration (Daw et al., 2005; Lee et al., 2014). Adaptive halting and early-exit methods instead vary the number of neural updates or the depth at which a prediction is emitted (Graves, 2016; Banino et al., 2021; Teerapittayanon et al., 2016). Iterative energy inference refines an output under a learned score (Belanger & McCallum, 2016; Belanger et al., 2017). These approaches vary inference effort, but leave a local question: *which semantic parts of competing actions need explicit evaluation, and which can reuse a learned summary?* Appendix A discusses the distinct computation mechanisms in detail.

We realize this idea with *Dual-Resolution Recursive Energy*. Each candidate action is supported by an AND/OR energy tree over local decision factors. Every internal subtree has two evaluation resolutions. Contracted evaluation uses a learned macro-energy that approximates the expanded value of that subtree, whereas expanded evaluation explicitly composes its constituent factors. In this operationalization, Contracted and fully expanded evaluation serve as operational analogues of the fast and deliberative endpoints; mixed-resolution evaluation lies between them and can combine both within one decision. Crucially, they are not separate controllers: both operate on the same recursive action energy, and different subtrees can use different resolutions within a single decision.

Energy provides the common interface between the two resolutions. The recursive structure combines required conditions and alternative support into comparable action energies; a macro-energy learns the value of the particular subtree it summarizes. Fast evaluation can therefore substitute a learned energy for explicit composition while preserving the role of the surrounding factors in the recursive root energy. This local correspondence is the central modeling choice. It also gives approximation error a direct role in action selection: a clear energy margin can accommodate a coarse evaluation, while a close comparison can call for finer resolution.

The framework couples this representation with learning and deliberation control. During wake, the model learns action preferences and records useful expansion traces. During sleep, those traces train macro-energies to reproduce expanded subtree values, consolidating expanded subtree evaluation into reusable compressed energies. Learned error-radius predictors estimate the associated contraction errors. At deployment, their propagated values determine whether the current action energies are sufficiently separated or further refinement is needed. Uncertainty here concerns the effect of finer evaluation on the action ranking. The controller can keep a well-separated decision compressed and expand selected subtrees when that ranking remains unresolved.

We evaluate this unified model across 17 tasks and layouts in BabyAI, MiniHack, and Overcooked-AI. It retains near-Full macro success with substantially less recursive evaluation. Comparisons with a learned global Fast/Slow controller, a learned nodewise gate, and randomized allocation examine the value of local mixed-resolution deliberation. On fixed held-out decision states, Ours reduces macro mean P50 latency relative to Full by 26.6%, 39.2%, and 57.5% across the three benchmark families, respectively, with 99.4%, 99.2%, and 100.0% macro action agreement. The reduction is consistent across tasks rather than driven by a few cases: Ours has lower measured P50 latency than Full on all 17 evaluated tasks and layouts. Component ablations and decision-level audits of the learned error estimates connect these results to the proposed mechanism.

Our contributions are threefold: **(1)** a local contract–expand representation in which learned subtree energies and explicit AND/OR evaluation occupy the same action-energy scale; **(2)** wake–sleep learning of reusable summaries and a margin-aware refinement rule that selects where to expand and when to stop, with a conditional winner-preservation result; and **(3)** evidence across three benchmark families that mixed-resolution deliberation retains near-Full task performance with less explicit expansion and lower decision latency than Full under a fixed-state timing protocol.

2 One Recursive Energy, Two Resolutions

2.1 Action-Conditioned Structure

At time t, the model receives a shared decision context x_t and encodes it as $z_t = q_\phi(x_t)$. Depending on the task interface, x_t may contain observations, bounded history, symbolic subgoals, and structured state features. For each candidate action $a \in \mathcal{A}$, an optional action-conditioned consequence representation u_t^a may additionally be derived from the shared context x_t (or its encoding z_t) together with the candidate action. The shared context and any such derived features form the information

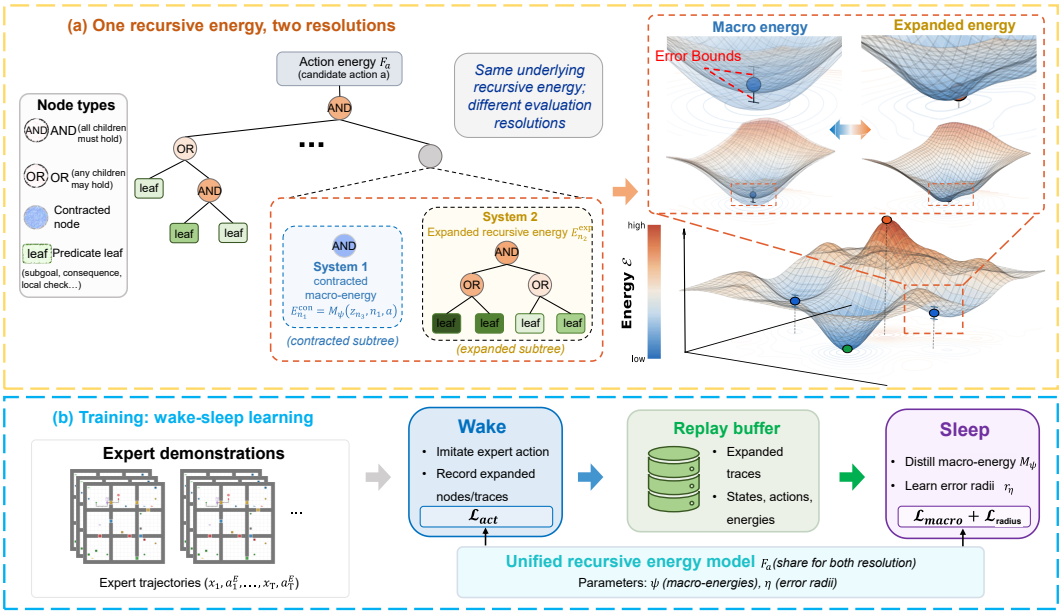


Figure 1: **Two energy resolutions within a unified decision model.** (a) Contracted macro-energies and expanded recursive energies represent the same underlying action-support structure at different resolutions. The schematic landscapes and energy gaps illustrate this correspondence and its approximation uncertainty. (b) Wake learns action preferences and records expansion traces; sleep distills expanded energies into reusable summaries and learns contraction-error predictors. Different subtrees can use different resolutions within one decision.

available to the action evaluator; how they are produced is separate from the contract–expand mechanism. Each action has a recursive AND/OR tree $T_a = (\mathcal{V}_a, r_a, \mathrm{ch}, \mathrm{op})$, with root r_a, leaf set $\mathcal{L}_a$, and internal-node set $\mathcal{I}_a$. Leaves encode grounded factors such as local executability, object relations, and progress toward the current subgoal. Internal nodes compose required conditions through AND and alternative support through OR. Lower energy indicates stronger support for an action under this factorization.

We assume that decision factors supporting each candidate action can be organized into an action-conditioned AND/OR tree. This recursive *scaffold* specifies how required and alternative factors compose, while the proposed contract–expand mechanism determines how that fixed structure is evaluated at different local resolutions. The scaffold can be derived from available task structure or induced from task descriptions and demonstrations. In our experiments, each scaffold is fixed before training and shared across recursive evaluators; repeated independent constructions are evaluated in Appendix L. Learning assigns state-dependent energies to its factors and trains compressed subtree evaluations. The external subgoal interface used for some long-horizon tasks is separate from the scaffold (Section 4.1). Figure 1 illustrates the representation and its learning procedure.

2.2 Expanded, Contracted, and Mixed Evaluation

A leaf predictor gives a soft truth value $s_\ell \in (0, 1)$ and energy $\epsilon_\ell = -\log s_\ell$. For candidate action a, its inputs are the shared context encoding z_t, action and leaf embeddings, and, when available, the derived action-conditioned consequence representation u^a_t. For child energies $v_1, \ldots, v_m$, the composition operators are

$$A_\wedge(v) = \sum_{i=1}^{m} v_i, \qquad A_\vee(v) = -\tau \log \sum_{i=1}^{m} e^{-v_i/\tau}. \tag{1}$$

Expanded evaluation recursively applies these operators to the subtree. In particular, an internal node has fully expanded value

$$E_n^{\mathrm{exp}} = A_{\mathrm{op}(n)}\big(\{E_c^{\mathrm{exp}} : c \in \mathrm{ch}(n)\}\big), \qquad E_\ell^{\mathrm{exp}} = \epsilon_\ell. \tag{2}$$

Contracted evaluation replaces an internal subtree with a learned macro-energy $E_n^{\mathrm{con}} = M_\psi(z_t, e_n, e_a)$. The macro-energy is trained toward the complete expanded value of the *same* subtree.

Let $S_a \subseteq \mathcal{I}_a$ contain the expanded internal nodes. It is frontier-consistent: expanding a node requires its internal ancestors to be expanded. Both resolutions are then expressed by one recursion:

$$E_n^{S_a} = \begin{cases} \epsilon_n, & n \in \mathcal{L}_a, \\ E_n^{\mathrm{con}}, & n \in \mathcal{I}_a \setminus S_a, \\ A_{\mathrm{op}(n)}\big(\{E_c^{S_a} : c \in \mathrm{ch}(n)\}\big), & n \in S_a. \end{cases} \tag{3}$$

Expanded and contracted evaluation therefore differ only in how the value of the same subtree is obtained: explicit recursive composition versus a learned surrogate. Both resolutions contribute to the same recursive action-energy calculation, allowing contracted and expanded subtrees to coexist within one decision. The root defines the action energy $F_a^{S_a} = E_{r_a}^{S_a}$. The choices $S_a = \varnothing$ and $S_a = \mathcal{I}_a$ yield contracted and fully expanded evaluation; intermediate sets yield local deliberation inside the same energy model.

For an AND node, each child supplies a required part of the action support, so its contribution enters additively. For an OR node, the soft minimum aggregates alternative supporting branches. A contracted node summarizes the fully expanded energy rooted there, including its descendant contributions. Expanding it exposes its children, which can themselves remain contracted or be refined further. This nested correspondence makes the granularity of deliberation adjustable throughout the tree.

Every resolution uses the same action distribution and decision rule:

$$\pi(a \mid x_t) = \frac{\exp(-F_a^{S_a}(x_t))}{\sum_{a'} \exp(-F_{a'}^{S_{a'}}(x_t))}, \qquad \hat{a}_t = \arg\min_a F_a^{S_a}(x_t). \tag{4}$$

Thus, additional thought changes the resolution at which action support is evaluated. Contracted subtrees provide reusable summaries, while expansion exposes their constituent factors. Appendix H specifies the remaining input and training conventions.

Figure 2 illustrates state-dependent deliberation: the same primitive action is selected using contracted, mixed, or expanded energy evaluation as the decision context changes. In the mixed state, explicitly evaluated branches and contracted neighboring subtrees contribute to the same root energy. Appendix F.2 explains the trajectory and the highlighted states.

3 Learning and Allocating Deliberation

3.1 Learning Fast Evaluation from Expanded Energies

Wake–sleep training connects the two resolutions. During wake, the current contract–expand evaluator produces mixed-resolution action energies for each expert decision point (x_t, a_t^E). Its objective is

$$\mathcal{L}_{\mathrm{wake}} = \mathcal{L}_{\mathrm{act}} + \mathcal{L}_{\mathrm{aux}}, \qquad \mathcal{L}_{\mathrm{act}} = -\log \pi(a_t^E \mid x_t). \tag{5}$$

Here $\mathcal{L}_{\mathrm{aux}}$ collects the enabled, weighted auxiliary objectives. Decision contexts and expansion traces enter a replay buffer.

During sleep, replayed traces supply expanded subtree targets. The model recomputes E_n^{exp} and trains E_n^{con} by regression to this stop-gradient target. It also predicts the local contraction gap $\Delta_n = |E_n^{\mathrm{exp}} - E_n^{\mathrm{con}}|$ with a nonnegative learned error estimate $\hat{r}_n = r_\eta(z_t, e_n, e_a)$. An asymmetric fitting loss penalizes underestimation more strongly than overestimation. The sleep objective combines these two terms:

$$\mathcal{L}_{\mathrm{sleep}} = \sum_{(t,a,n)\in\mathcal{R}} \ell_{\mathrm{macro}}(E_n^{\mathrm{con}}, \mathrm{sg}(E_n^{\mathrm{exp}})) + \lambda_{\mathrm{radius}}\mathcal{L}_{\mathrm{radius}}. \tag{6}$$

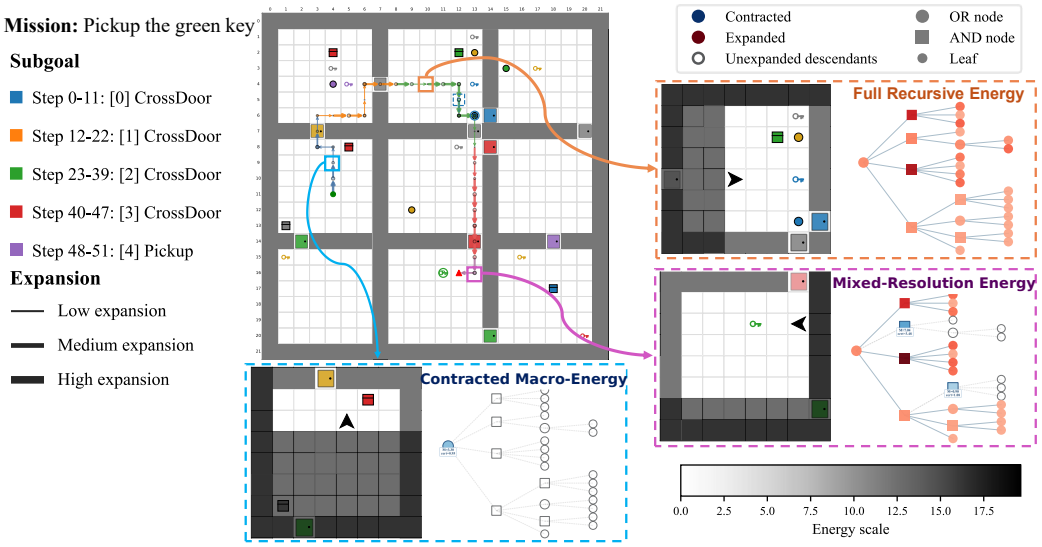


Figure 2: **Adaptive deliberation within one recursive energy model.** Three states in a BabyAI rollout select `forward` using contracted, mixed, and expanded evaluation. Trajectory line width indicates the amount of expansion at each decision. Blue nodes are contracted, red nodes are expanded, and hollow descendants remain implicit.

Alternating wake and sleep learns task behavior and consolidates expanded computation into the fast evaluation of its underlying substructures. Auxiliary objectives and environment-specific training settings are given in Appendices H and B. Deployment uses the learned energies and error-radius predictors directly.

Learning decision-relevant error estimates. The error-radius head is trained on the same replayed subtree pairs as the macro-energy. With $\bar{\Delta}_n = \mathrm{sg}(\Delta_n)$, its loss is

$$\mathcal{L}_{\mathrm{radius}} = \sum_{(t,a,n)\in\mathcal{R}} w_n \left[\lambda_{\mathrm{under}}(\bar{\Delta}_n - \hat{r}_n)_+ + \lambda_{\mathrm{over}}(\hat{r}_n - \bar{\Delta}_n - \delta)_+\right]. \tag{7}$$

Here $(v)_+ = \max(v, 0)$, $\lambda_{\mathrm{under}} > \lambda_{\mathrm{over}}$, $\delta \geq 0$ is a fitting slack, and w_n can emphasize decision-critical nodes. These are training-loss parameters. At inference, the learned predictions enter the winner-separation test directly. They are learned estimates, not unconditional upper bounds. Recomputing expanded targets from replayed contexts keeps the compressed evaluation aligned with the current recursive model as action learning progresses.

3.2 Expanding Where the Decision Needs More Resolution

All action trees start contracted. For each partial tree, learned estimates of local contraction error propagate upward: leaves contribute zero, an unresolved internal node contributes $\hat{r}_n$, and an expanded node sums child values for AND or takes their maximum for OR. Write $B_a^{S_a} = B_{r_a}^{S_a}$ for the resulting root value. This propagation translates local contraction uncertainty into the action-energy scale. Explicitly, the propagated value is

$$B_n^{S_a} = \begin{cases} 0, & n \in \mathcal{L}_a, \\ \hat{r}_n, & n \in \mathcal{I}_a \setminus S_a, \\ \sum_{c\in\mathrm{ch}(n)} B_c^{S_a}, & n \in S_a,\ \mathrm{op}(n) = \wedge, \\ \max_{c\in\mathrm{ch}(n)} B_c^{S_a}, & n \in S_a,\ \mathrm{op}(n) = \vee. \end{cases} \tag{8}$$

The sum reflects the additive sensitivity of AND composition; the maximum follows from the soft minimum's 1-Lipschitz property in the infinity norm.

Let $a^\star = \arg\min_a F_a^{S_a}$ be the current winner. Inference stops when its predicted interval is separated from every competitor:

$$F_a^{S_a} - F_{a^\star}^{S_{a^\star}} > B_a^{S_a} + B_{a^\star}^{S_{a^\star}}, \qquad a \neq a^\star. \tag{9}$$

When more resolution is needed, the unresolved competitors are

$$\mathcal{C} = \left\{ a \neq a^\star : F_a^{S_a} - F_{a^\star}^{S_{a^\star}} \leq B_a^{S_a} + B_{a^\star}^{S_{a^\star}} \right\}. \tag{10}$$

Let $\mathcal{F}_a^{S_a}$ be the exposed, contracted internal frontier. Eligible expansion pairs are

$$\mathcal{Q} = \left\{ (a, n) : a \in \{a^\star\} \cup \mathcal{C},\ n \in \mathcal{F}_a^{S_a} \right\}. \tag{11}$$

These definitions connect the two scales of allocation. At the action level, the controller focuses on the current winner and alternatives that can still challenge it according to their predicted intervals. Within each eligible action, the frontier enforces the recursive dependencies, exposing detail only through its parent subtree. The controller selects the eligible pair (a, n) with the largest propagated error estimate,

$$\text{priority}_a(n) = B_n^{S_a}, \tag{12}$$

expands that node, and updates affected energies and propagated estimates. Because the reported experiments treat each explicit node expansion as one structural evaluation unit, refinement directly prioritizes the unresolved frontier node with the largest predicted contribution to action-energy uncertainty. The winner and eligible frontier are recomputed after refinement. This procedure concentrates explicit evaluation on unresolved action comparisons; the final action follows Eq. (4). Complete stopping conditions and pseudocode are given in Algorithm 1.

This stopping rule couples the predicted approximation error to the decision being made. A large action margin can accommodate uncertain local summaries; a narrow margin can justify refinement even when those summaries are fairly accurate. Expanding a node reveals finer-grained subtree energies and error estimates, which can change both the current winner and the next location selected for inspection. The resulting deliberation resolution therefore emerges separately for each decision from the learned model and the exposed tree structure.

Conditional winner preservation. The learned radii provide deployment-time predictions, not guaranteed coverage. Define local undercoverage $q_n = (\Delta_n - \hat{r}_n)_+$ and propagate it with the same sum/max rules to obtain $R_a^{S_a}$. The recursive operators give the bound

$$|F_a^{\text{exp}} - F_a^{S_a}| \leq B_a^{S_a} + R_a^{S_a}. \tag{13}$$

Proposition 1 (Conditional preservation of the Full winner)**.** *Suppose Eq.* (9) *stops with winner $a^\star$, and let $\gamma_a = F_a^{S_a} - F_{a^\star}^{S_{a^\star}} - B_a^{S_a} - B_{a^\star}^{S_{a^\star}} > 0$ for each $a \neq a^\star$. If $R_a^{S_a} + R_{a^\star}^{S_{a^\star}} < \gamma_a$ for every competitor, then $F_{a^\star}^{\text{exp}} < F_a^{\text{exp}}$ for all $a \neq a^\star$. In particular, complete coverage ($\Delta_n \leq \hat{r}_n$ at unresolved nodes) suffices.*

The residuals are observable only in post-hoc analysis; inference stops using the predicted B values. The proposition also allows some local undercoverage when it fits within the remaining action margin. Appendix J proves the propagation bound and winner-preservation result.

4 Experiments

We test whether dual-resolution evaluation preserves task performance with less explicit expansion, whether the change in resolution yields measured decision-time differences, and how approximation errors interact with the action margin. Representation ablations and transfer tests are reported in Appendices D and E.

4.1 Tasks and Deliberation Controls

Benchmarks. We evaluate six BabyAI tasks (Chevalier-Boisvert et al., 2019), eight MiniHack tasks (Samvelyan et al., 2021), and three Overcooked-AI layouts (Carroll et al., 2019); Overcooked-AI uses Success@2, the fraction of rollouts delivering at least two soups within 100 joint-action

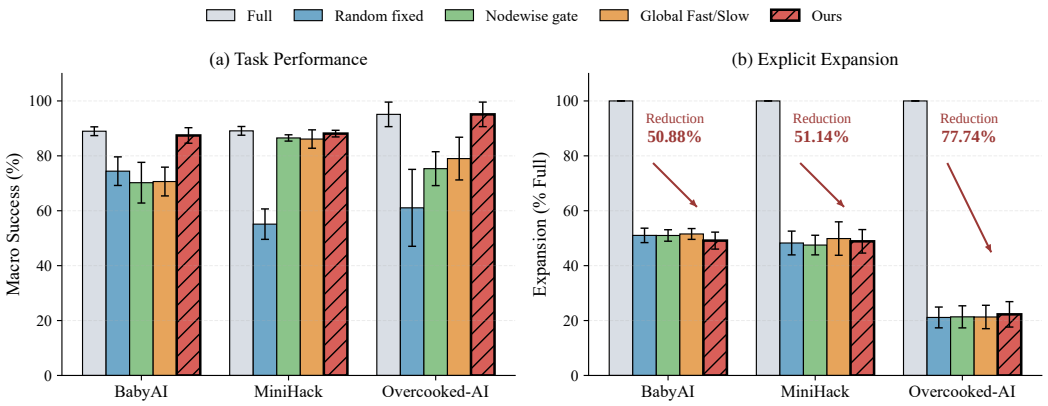


Figure 3: **Strong task performance with local mixed-resolution deliberation.** Left: benchmark macro success. Right: explicit expansion relative to Full. Task or layout metrics are averaged within each run; bars and error bars show mean±SD across six training seeds.

steps. For BabyAI and MiniHack, all compared low-level methods share the same upstream subgoal interface and shared decision context x_t; when used, one-step action-conditioned consequence features u_t^a are derived from that context for each candidate action and supplied identically to the matched methods. Overcooked-AI uses neither the subgoal interface nor consequence features. These upstream inputs are separate from contract–expand inference; full interfaces, supervision, and construction details are in Appendix B.

Controls. Full recursive, Random fixed, and Ours share the same trained checkpoint and scaffold; they differ only in inference-time expansion. Nodewise gate and Global Fast/Slow provide learned local- and decision-level allocation controls. Tunable controls operate near Ours' average structural budget; details are in Appendix B.10.

Evaluation. We report mean±SD over six training seeds with 100 closed-loop episodes per task or layout; selected method differences additionally use paired 95% percentile-bootstrap intervals (Appendix C). Expansion is normalized by Full, and decision latency follows the fixed-state protocol below.

4.2 Task Performance with Less Recursive Evaluation

Figure 3 shows that Ours retains near-Full macro success with substantially less recursive evaluation. On BabyAI, it achieves 87.42% success versus Full's 88.97% while expanding 49.12% of the scaffold; on MiniHack, 88.10% versus 89.10% at 48.86% expansion; and on Overcooked-AI, it matches Full at 95.11% while expanding only 22.26%. Relative to the same-checkpoint Full evaluator, Ours differs in macro success by only 1.55, 1.00, and 0.00 percentage points while reducing explicit expansion by 50.88%, 51.14%, and 77.74% on BabyAI, MiniHack, and Overcooked-AI, respectively.

At similar structural budgets, the benchmark-level difference from the strongest non-Full control is $+13.00$ percentage points on BabyAI versus Random fixed (paired 95% bootstrap CI $[+9.50, +16.25]$), $+1.58$ points on MiniHack versus Nodewise gate ($[-1.02, +4.50]$), and $+16.11$ points on Overcooked-AI versus Global Fast/Slow ($[+5.00, +28.39]$). Thus the paired interval is clearly positive on BabyAI and Overcooked-AI, while the smaller MiniHack difference is not separated from zero at this sample size. On MiniHack, Nodewise gate already reaches 86.52% and Full reaches 89.10%, while Ours achieves 88.10% with 48.86% expansion. The same-checkpoint Full comparison isolates inference-time resolution, while the matched-budget controls compare ways to allocate expansion.

The main BabyAI scaffolds are LLM-generated and manually audited; MiniHack and Overcooked-AI scaffolds are LLM-generated with structural consistency checks (Appendix B.6). Holding each scaffold fixed across seeds and recursive controls separates scaffold construction from their within-

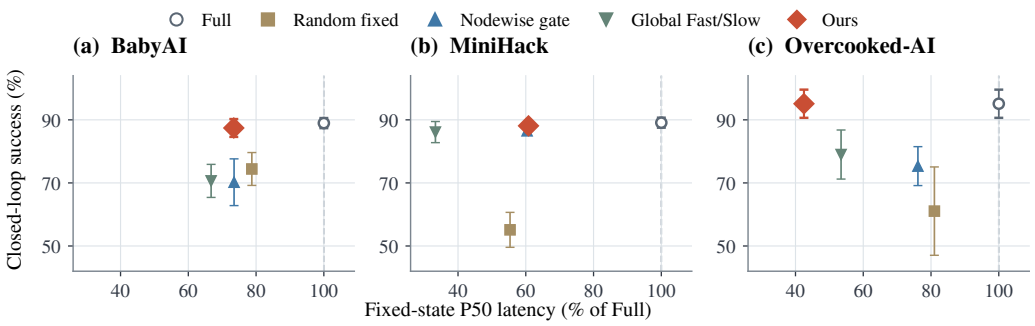


Figure 4: **Closed-loop success and fixed-state decision latency.** Success is the benchmark macro mean with ± 1 SD across six training seeds from closed-loop rollouts. Latency is the macro mean of task-level P50 values on identical held-out expert states, expressed relative to Full within each benchmark. Lower latency and higher success are better. The plot includes all five methods with corresponding timing data.

task comparisons. The inference ablation in Appendix D further separates *when to stop* from *where to expand*. With the same learned radii and stopping rule but a fixed randomized frontier order, RF-order + Stop expands 64.16%, 69.58%, and 49.10% of Full on BabyAI, MiniHack, and Overcooked-AI, versus Ours' 49.12%, 48.86%, and 22.26% at comparable benchmark-level success. In the reported No-action ablation, removing action-level learning while retaining the scaffold and predicate supervision yields 0% success under that training setup. NoLeaf retains the tree and action objective while learning latent leaf factors without direct predicate labels; its performance is task-dependent. Independently induced `UnblockPickup` scaffolds, each followed by a new training run, reach $82.0 \pm 4.4\%$ success versus $85.67 \pm 6.35\%$ for the reference scaffold (Appendix L). The No-action result shows that the supplied tree and predicate targets do not suffice for learning task actions in that training setup; NoLeaf tests the need for direct predicate labels, and the independent constructions test one task's sensitivity to the scaffold. The decision-level audit in Section 4.4 examines predicted contraction errors; complete task results appear in Appendix C.

On retained BabyAI `BossLevel` missions using the training skill repertoire, the `UnblockPickup`-trained controller expands 32.52 of 34 nodes, up from 18.84 on `UnblockPickup`, without retraining or recalibration. It attains 68.33% success versus Full's 68.67% under transfer (Appendix E), showing that evaluation shifts toward finer resolution in more complex task contexts.

4.3 Measured Speed and Adaptive Resolution

We time low-level action selection on held-out expert decision states across BabyAI, MiniHack, and Overcooked-AI, holding the inputs fixed across methods within each task or layout. Relative to Full, Ours reduces macro mean P50 latency by 26.6%, 39.2%, and 57.5%, respectively, while achieving 99.4%, 99.2%, and 100.0% macro action agreement with Full (Appendix Figure 13). Figure 4 places these latency reductions alongside benchmark success and the timed controls. This fixed-context comparison isolates recursive action-selection time and complements the closed-loop results, in which methods generate their own trajectories; it excludes high-level planning, environment steps, and generation of the precomputed consequence features. Appendix G provides the timing protocol and per-task latencies for all controls. Direct or global policies can have lower latency with lower task success; the Full comparison tests the decision-time effect of local contraction within the same trained recursive model, together with its action agreement.

Training dynamics and a uniform-budget sweep in Appendix F further connect learned contraction to state-dependent deliberation.

4.4 Decision-Level Reliability of Learned Error Estimates

We audit frozen checkpoints under the closed-loop protocol used for task performance. At each audited state, partial and Full evaluation receive the same observation context. Instrumented replay is

Table 1: **Reliability of learned error radii and early stopping.** Early stopping uses all audited decisions; other rates condition on early stops. Macro mean±SD is over checkpoint replicates.

Benchmark	ES (%)↑	RMV (%)↑	RealV (%)↑	Disagree (%)↓
BabyAI	78.08 ± 6.17	83.67 ± 4.48	90.54 ± 2.64	6.09 ± 0.92
MiniHack	85.07 ± 25.14	93.16 ± 6.58	96.30 ± 4.71	2.81 ± 3.48
Overcooked-AI	99.38 ± 0.57	98.93 ± 0.64	99.56 ± 0.54	0.06 ± 0.08

verified against the production implementation before diagnostics are computed. This audit connects the learned local error estimates to the action they help select.

Metrics and their denominators. The audit connects three levels: local contraction error, propagated action-energy uncertainty, and the resulting action choice. *Early stopping* denotes the fraction of audited decisions satisfying the predicted winner-separation rule before full expansion; it does not imply that every learned radius covers its realized error. The remaining decision-level rates are measured *among early-stopped decisions*, with early-stop winner $a^\star$ and positive separation slack γ_a as defined in Section 3.2. *Residual-margin validity* checks whether $R_a^{S_a} + R_{a^\star}^{S_{a^\star}} < \gamma_a$ for every competitor, using the propagated local undercoverage from Section 3.2. It evaluates the sufficient winner-preservation condition. *Realized-margin validity* instead uses the observed partial-to-Full energy deviations $d_a = |F_a^{\exp} - F_a^{S_a}|$ and checks

$$F_a^{S_a} - F_{a^\star}^{S_{a^\star}} > d_a + d_{a^\star}, \qquad \forall a \neq a^\star. \tag{14}$$

Early-stop Full-winner disagreement counts early-stopped decisions whose early-stop winner differs from the Full winner on the same decision context. The two validity metrics test sufficient conditions; disagreement measures the observed action outcome.

By comparison, *strict frontier coverage* counts unresolved nodes whose local gap satisfies $\Delta_n \leq \hat{r}_n$. The training-dynamics *violation rate* checks local coverage on held-out expert contexts from the training–validation split (Appendix F.4). These node-level checks and the decision-level audit have distinct denominators and evaluation contexts. Their relationship is mediated by propagation and the current action margin.

Reliability through the winner margin. Table 1 shows how often the learned error radii support an early stop and how the predicted margin rule behaves against Full at the same decision context. Among early-stopped decisions, Full-winner disagreement is 6.09%, 2.81%, and 0.06% on BabyAI, MiniHack, and Overcooked-AI, respectively. Most early stops satisfy the post-hoc margin-validity tests; their sufficient conditions can fail even when the action agrees with Full. These decision-level disagreements do not directly measure episode failure: the success difference from Full is 1.55, 1.00, and 0.00 percentage points across the three benchmarks, respectively. Appendix F.1 analyzes the severity of undercoverage relative to the remaining action margin.

The residual-severity analysis further connects the empirical audit to the winner-preservation condition. Among audited early stops, no observed Full-winner change occurs when the realized undercoverage remains below the available action margin ($\rho < 1$); observed changes are confined to the $\rho \geq 1$ region, where the sufficient condition no longer guarantees the Full winner (Appendix F.1). Thus, failures of strict local coverage need not change the selected action when the remaining decision margin is sufficiently large.

5 Conclusion

Dual-Resolution Recursive Energy lets learned subtree summaries and explicit AND/OR evaluation coexist within one action energy. Margin-aware refinement exposes factors when the predicted contraction error can still affect the selected action. Across BabyAI, MiniHack, and Overcooked-AI, the model retains near-Full task performance with less explicit expansion and lower fixed-state decision latency than Full. Expansion is our operational proxy for the agent's extent of deliberation; the winner-preservation result is conditional on error coverage or sufficient remaining action margin.

AI USE STATEMENT

Generative AI tools assisted with language editing, manuscript organization, LaTeX preparation, feedback on methodological presentation, interpretation of results, and internal consistency checks. The authors reviewed the AI-assisted outputs. LLM-based components of the experimental methodology are described in the paper and appendix. The authors take responsibility for the final content, experimental analysis, and scientific claims.

REPRODUCIBILITY STATEMENT

Algorithm 1 specifies deployment-time inference. Appendix H specifies additional model and training conventions; Appendix J gives the proofs. Appendix B specifies task interfaces, supervision, evaluation protocols, hyperparameters, and baselines. Complete task results, component ablations, and fixed-state expansion and wall-clock measurements appear in Appendices C, D, and G. Appendix L documents scaffold construction; Appendices M and N give the construction and planner prompts.

REFERENCES

Zergham Ahmed, Kazuki Irie, Joshua B. Tenenbaum, Christopher J. Bates, and Samuel J. Gershman. Learning abstractions for hierarchical planning in program-synthesis agents, 2026. URL https://arxiv.org/abs/2602.00929.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

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

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

Mohak Bhardwaj, Sanjiban Choudhury, Byron Boots, and Siddhartha Srinivasa. Leveraging experience in lazy search. In *Proceedings of Robotics: Science and Systems*, 2019. doi: 10.15607/RSS.2019.XV.050.

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

Frederick Callaway, Sayan Gul, Paul M. Krueger, Thomas L. Griffiths, and Falk Lieder. Learning to select computations. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 776–785, 2018. URL https://auai.org/uai2018/proceedings/papers/269.pdf.

Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In *Advances in Neural Information Processing Systems*, volume 32, pp. 5175–5186, 2019.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019.

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

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

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

Jonathan St BT Evans and Keith E Stanovich. Dual-process theories of higher cognition: Advancing the debate. *Perspectives on psychological science*, 8(3):223–241, 2013.

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

Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pp. 346–355, 2012.

Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal. The Wake-Sleep algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

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

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

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

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

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

Wenjie Shen, Boyang Li, Chao Yang, and Shuang Li. Deliberate-when-needed: Flow-Reasoner for neuro-symbolic continuous thought. In *Proceedings of The 29th International Conference on Artificial Intelligence and Statistics*, 2026.

Steven A. Sloman. The empirical case for two systems of reasoning. *Psychological Bulletin*, 119(1): 3–22, 1996. doi: 10.1037/0033-2909.119.1.3.

Ron Sun, Paul Slusarz, and Chris Terry. The interaction of the explicit and the implicit in skill learning: A dual-process approach. *Psychological Review*, 112(1):159–192, 2005. doi: 10.1037/0033-295X.112.1.159.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

A Related Work

A computational interpretation of dual-process accounts. Dual-process theories distinguish relatively automatic processing from deliberative reasoning (Sloman, 1996; Evans, 2003; Evans & Stanovich, 2013). Computational accounts include interacting implicit and explicit systems (Sun et al., 2005) and arbitration between habitual and planning-based controllers (Daw et al., 2005; Dolan & Dayan, 2013; Lee et al., 2014). Our framework instantiates fast and deliberative evaluation as two resolutions of one recursive decision representation. A contracted macro-energy is trained on the expanded value of the subtree it replaces, making their relationship explicit at the level of local decision factors. Flow-Reasoner (Shen et al., 2026) also invokes symbolic reasoning selectively, but studies continuous-time behavioral prediction and triggers rule-based updates at selected times. Our setting instead selects primitive actions by changing which subtrees of one action-conditioned energy are explicitly evaluated within each decision.

Recursive resolution as a computation variable. Energy-based learning supplies a shared scoring language for prediction, planning, and action selection (LeCun et al., 2006; Belanger & McCallum, 2016; Belanger et al., 2017; Boney et al., 2020; Liu et al., 2021). Structured reasoning provides intermediate factors and compositional dependencies (Manhaeve et al., 2018; Yao et al., 2023). In our construction, each subtree supports an explicit recursive value and a compressed estimate on the same energy scale. Changing an expansion set changes which factors are evaluated while retaining the recursive action-scoring rule. This is the representation-level connection between the two modes.

Adaptive computation time and energy resolution. ACT (Graves, 2016) learns how many recurrent state updates to perform for each input, using a halting unit and a ponder-cost penalty. PonderNet (Banino et al., 2021) learns a conditional halting distribution over repeated applications of a neural step function, with regularization toward a geometric prior. Early-exit networks similarly choose the depth at which to emit a prediction (Teerapittayanon et al., 2016). In these formulations, adaptation changes the duration or depth of a sequence of neural computations. Our setting is primitive-action control through a recursive energy representation. Adaptation changes which semantic subtrees are summarized by macro-energies and which are composed from explicit factors, with both resolutions expressed on the same action-energy scale. The correspondence is between a subtree summary and that subtree's expanded value; both resolutions can coexist within one action comparison. ACT and PonderNet provide relevant context for adaptive effort, while the global and nodewise controls in our experiments compare allocation within the shared task interfaces.

Selecting decision-relevant computations. Metareasoning studies how to allocate computation among candidate decisions, including when to stop and which computation to perform (Hay et al., 2012; Callaway et al., 2018). Learned lazy search prioritizes graph-edge evaluations during planning (Bhardwaj et al., 2019). Our unit of refinement is an action-conditioned semantic subtree: a learned macro-energy and its expanded recursive value occupy the same action-scoring scale. Propagated estimates of their discrepancy guide which subtrees to expand according to the current winner margin, so coarse and fine evaluations can coexist within a decision.

Learning reusable energy summaries. Wake–sleep learning (Hinton et al., 1995) motivates alternating behavior learning with consolidation. Here, expanded subtree energies train their contracted counterparts and local error predictors. The common energy scale makes these errors meaningful for comparing candidate actions, linking dual-resolution representation to selective refinement and margin-aware stopping.

B Implementation and Experimental Setup Details

B.1 Environment Interfaces

BabyAI and MiniHack use subgoal-conditioned primitive-action control; Overcooked-AI uses direct primitive-action coordination. Context features follow each task interface, while the recursive energy formulation and margin-aware evaluator are shared across families.

B.2 High-Level Task Interface

The proposed method operates on primitive-action decisions after a decision context x_t has been formed. For BabyAI and MiniHack, we therefore separate long-horizon task decomposition from the low-level adaptive inference problem studied in this paper. A shared symbolic subgoal interface supplies the active subgoal to every compared low-level controller from an admissible vocabulary fixed in advance. This upstream module is not part of the proposed contract–expand evaluator and does not construct or evaluate the action-conditioned AND/OR scaffold.

In BabyAI, the shared planner uses the full symbolic environment state to produce the high-level plan. The low-level controllers receive only the same coordinate-free projection of the active subgoal together with their runtime decision context. Target coordinates that may appear in planner records or offline annotation are excluded from low-level model inputs during both training and inference. MiniHack uses an analogous coarse stage-level decomposition. Overcooked-AI uses direct primitive-action coordination and requires no high-level planner.

This decomposition is an experimental choice for isolating the contribution of adaptive structured action evaluation rather than a requirement of the method. Jointly learning long-horizon exploration, memory, hierarchical task decomposition, and low-level recursive evaluation would introduce a substantially larger learning problem orthogonal to the contract–expand mechanism. With 400 successful demonstrations per environment and the available training budget, we therefore keep the upstream decomposition fixed and compare low-level evaluators under identical supplied subgoals. For tasks that require no multistage decomposition, the interface directly supplies the original task objective.

The LLM components use GPT-5.4 with temperature 0.2. Planning prompts and the output schema appear in Appendix N.

B.3 Dataset Construction and Supervision

Training uses successful demonstration trajectories aligned at pre-action decision points. BabyAI and MiniHack each use 400 expert demonstrations per environment. MiniHack demonstrations are collected with a dedicated scripted expert designed for the coarse stage-level task interface. The exact number of low-level action steps varies with task and trajectory length.

Each training example provides primitive-action supervision. BabyAI and MiniHack additionally use the active symbolic subgoal without global target coordinates as a model input. The default experimental configurations use predicate-level labels for semantic leaf grounding when such labels are available. These labels provide optional auxiliary supervision rather than a requirement of the recursive representation: the NoLeaf ablation removes direct predicate supervision while retaining the same scaffold, action objective, and contract–expand inference. LLM-generated predicate annotations are offline auxiliary targets; the deployed low-level controller predicts its leaf values from the available decision context without querying the predicate labeler.

Overcooked-AI uses the corresponding action-supervised training interface but does not introduce LLM-derived subgoals or future-prediction supervision. Only player 0 is learned. Demonstrations for `cramped_room` and `asymmetric_advantages` are generated by two official `GreedyHumanModel` agents. For `large_room`, demonstrations are generated by a coordinated joint planner with server and onion-runner roles; the learned player-0 policy takes the server role. Partner policies and the evaluation protocol are specified in Appendix B.9.

B.4 Observation and Structured State Representation

Within each environment, Ours and the mechanism-matched baselines receive the same observable decision context. BabyAI uses the default egocentric $7 \times 7 \times 3$ partial observation together with bounded observation/action history and locally observable structured features. MiniHack follows the same principle using its local symbolic observation and observable inventory or command state. Overcooked-AI uses its environment-specific local coordination state.

The structured observation features are derived from runtime-observable information, including agent self-state, local affordances, observable inventory or carried-object status, and visibility/history

indicators. They complement the shared task context described above. All matched methods within each benchmark receive the same runtime decision interface.

B.5 Optional Action-Conditioned Consequence Features

The contract–expand formulation does not require a particular dynamics model. In BabyAI and MiniHack, the shared decision context x_t is first encoded as $z_t = q_\phi(x_t)$. For each candidate primitive action a, we optionally derive a one-step action-conditioned consequence representation u_t^a from (z_t, a). Thus, x_t denotes information shared across candidate actions, whereas u_t^a denotes action-specific predicted consequences derived from that shared context.

This representation plays the role of action-conditioned imagination commonly supplied by a world model or transition model. In our implementation, a fixed offline learned Python transition function takes the currently observable symbolic context and candidate action, simulates one step, and encodes its predicted local effects, following the code-based world-model approach of TheoryCoder-2 (Ahmed et al., 2026). Whenever consequence features are used, the same u_t^a are supplied identically to Ours and the mechanism-matched controls. Overcooked-AI uses no such features and applies the same contract–expand evaluator directly to its shared coordination-state context.

B.6 Action-Conditioned Recursive Scaffolds

Scaffolds are constructed and checked before training, then fixed across the six training seeds and all recursive allocation controls for each task. The main BabyAI scaffolds are generated by an LLM and manually audited before being fixed; Appendix K illustrates one such scaffold. MiniHack and Overcooked-AI scaffolds are generated by an LLM from task semantics, action and predicate vocabularies, and demonstrations, followed by structural consistency checks. Their predicate vocabularies cover the navigation, inventory, command, and coordination factors appropriate to each environment. In Overcooked-AI, `asymmetric_advantages` reuses the 54-node `cramped_room` scaffold; `large_room` uses a separate 66-node scaffold. Independent construction repetitions are evaluated in Appendix L.

B.7 Training Objectives

The experimental instantiations use action imitation as the primary wake objective. For BabyAI and MiniHack, the default configuration additionally uses fully expanded action supervision and predicate-level grounding. The NoLeaf ablation isolates the contribution of the predicate-grounding term. With the fixed programmatic consequence interface, training focuses on action learning, grounding, macro-energy distillation, and error-radius fitting.

BabyAI and MiniHack use a warmup stage, an expanded-evaluation pretraining stage, and subsequent wake–sleep optimization. Wake updates are frozen after the configured epoch while sleep continues to refine macro-energies and predicted error radii. Overcooked-AI uses the same core action-learning and contract–expand training principle without planner- or future-specific components. The principal hyperparameters are summarized below.

B.8 Training Hyperparameters

Table 2 summarizes the default training and data settings. Table 3 reports the loss weights, and Table 4 reports model, contract–expand inference, and sleep-stage hyperparameters. The tables summarize the principal settings used in the main experiments; most are shared across environments.

Table 2: 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
History length	6
Future horizon (when used)	1

Table 3: Principal loss weights and error-radius fitting coefficients.

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_{radius}	1.0
$\lambda_{\text{wake_expanded_aux}}$	0.1
λ_{under}	5.0
λ_{over}	0.25

Table 4: Principal model, inference, and sleep-stage settings.

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
Maximum explicit expansions	66
Sleep steps per epoch	200
Replay buffer capacity	20000
Sleep batch size	32
Leaf class-balance weighting	Enabled

B.9 Inference Protocol and Metrics

Our primary closed-loop metric is task success. For BabyAI and MiniHack, an episode is successful when the environment-defined task objective is completed within the evaluation horizon. For Overcooked-AI, we report Success@2: an episode is successful if at least two soups are delivered within 100 primitive joint-action steps. At each step, the learned player-0 action and the scripted player-1 action are executed simultaneously. A rollout stops earlier if the environment returns `done`; otherwise it runs for 100 steps. Each training seed and evaluation bank uses 100 rollouts. Unless otherwise stated, success rates are computed over 100 closed-loop evaluation episodes for each of six independent training runs.

The Overcooked-AI evaluation partner is fixed and is not trained. In `cramped_room` and `asymmetric_advantages`, player 1 is the official `GreedyHumanModel`. In `large_room`, player 1 is `RoleAwareAgent(onion_runner)` and the learned player 0 takes the server role. The partner observes the full environment state and selects actions using the official medium-level planner. Partner tie-breaking randomness is seeded per rollout and shared across methods within each evaluation bank, providing a reproducible partner for each test rollout.

Adaptive deliberation is measured by the number of explicitly expanded recursive nodes and by the expansion ratio relative to the corresponding Full tree. We additionally report measured decision latency and fixed-state action agreement with Full where appropriate. Full uses the same recursive formulation with contraction disabled, evaluating the complete recursive structure. Action accuracy, predicate accuracy, and learned-radius diagnostics are used to analyze individual components rather than as primary benchmark metrics.

B.10 Baseline Implementations

We compare with a standard behavior-cloning reference and baselines that select different degrees or locations of recursive expansion. Ours does not target a preset expansion ratio; its average expansion level follows from predicted margin-based stopping. For baselines with tunable expansion, we

configure each task or layout to approximate Ours's average expansion ratio as closely as the baseline permits. Matching uses expanded-node counts relative to Full; task performance and decision latency are evaluated separately.

Flat-BC. Flat-BC directly maps the shared observable decision context to primitive actions using a standard behavior-cloning policy head. It provides a non-recursive behavior-cloning reference.

Full recursive. For each training seed, Full and Ours use the same trained recursive-energy checkpoint; only the inference-time expansion rule differs. Full explicitly evaluates every internal node of the action-conditioned scaffold at each decision, providing the fully expanded reference for comparisons of task performance, explicit expansion, and decision latency.

Random fixed. Random fixed uses a state-independent expansion order. It uses the same trained recursive-energy checkpoint and scaffold as Ours and matches Ours's average structural budget. Expandable nodes follow a fixed randomized frontier ordering throughout evaluation. This baseline tests whether a fixed frontier order at a comparable expansion level can match state-dependent refinement.

Nodewise gate. Nodewise gate is a learned local-allocation baseline without propagated error radii or winner separation. We freeze the recursive scoring model and train a separate gate to rank expandable action–node pairs according to their usefulness for recovering the winner under fully expanded evaluation. For a candidate node, its training utility measures the increase in log-probability assigned to the full-resolution winner after expanding the required ancestor path, normalized by the corresponding additional expansion cost. A listwise cross-entropy objective trains the gate to reproduce the relative ordering of these winner-aware utilities, with a small auxiliary SmoothL1 term toward the local expanded–contracted energy-gap-per-cost target.

At inference time, the trained gate predicts scores for all action–node pairs once at the beginning of each decision. At each refinement step, the controller selects the highest-scoring pair on the currently reachable frontier. If its score exceeds the threshold, the node is expanded and the reachable frontier is updated; otherwise, refinement stops. The gate scores remain fixed during this decision. The threshold is configured per task to operate at an average structural budget close to Ours. The final action is selected from the resulting partially expanded recursive energy.

Global Fast/Slow. Global Fast/Slow provides a learned decision-level arbitration baseline with no intermediate recursive refinement. Its Fast branch is a directly learned action head over the shared encoded decision context and action-conditioned consequence features; it does not access predicate scores, recursive-tree states, macro- energies, or predicted error radii. The Fast head is trained with expert action supervision together with distillation from the frozen fully expanded recursive scorer. Its Slow branch performs complete recursive evaluation, so each decision uses either zero explicit recursive expansion or the full tree. This comparison tests whether a learned decision-level Fast/Slow switch under the shared task interface can match local mixed-resolution refinement.

After training the Fast branch, we freeze both the recursive scorer and the Fast head and train a separate global gate to predict whether invoking Slow is useful for the current decision. The gate receives only Fast-side information, including the shared context representations, Fast action logits, and global confidence statistics such as entropy, top-two margin, and maximum action probability. Slow outputs are used only to construct training targets and are not available to the gate at inference time. Gate targets identify states for which Slow corrects or meaningfully improves upon the Fast prediction, and the gate is optimized with a class-balanced binary classification objective.

At inference time, the gate produces one Slow probability per decision. If it exceeds a task-specific threshold, the complete recursive tree is evaluated; otherwise the Fast action is returned directly. The threshold is configured to operate at an average structural budget close to Ours, independently of task success. Thus, Global Fast/Slow tests whether coarse decision-level arbitration between no recursive evaluation and full recursive expansion can match the benefits of local mixed-resolution refinement.

B.11 Compute Resources

The experiments were conducted primarily on NVIDIA RTX 3090 GPUs. Individual training runs were executed on a single GPU, while environments and runs were parallelized when resources were available. The reported main results use six independent training runs with different random seeds.

The decision-time latency measurements reported in the main text were collected separately under a fixed-state benchmark protocol. Training-time compute is therefore reported only at the hardware/-configuration level rather than used as a comparative metric.

C Complete Benchmark Results

Table 5 supplies the exact values underlying the main success–expansion figure. Table 6 reports paired benchmark-level success differences and 95% percentile-bootstrap intervals. Tables 7– 9 give task-level success, expanded nodes, and expansion relative to Full. Flat-BC is the additional non-recursive reference; structural expansion is not applicable to that policy.

In Tables 5–9, the Ours row is shaded for readability.

The task-level comparisons show where local refinement is especially useful: multi-stage object interaction in BabyAI, command- and inventory-sensitive decisions in MiniHack, and coordination in Overcooked-AI. The results also retain task-specific variation that is averaged in the benchmark summaries.

Table 5: Exact benchmark summaries for Figure 3. Task metrics are averaged within each run before computing mean±SD across six training seeds. Expansion is relative to Full.

Benchmark	Method	Success (%) ↑	Expansion (% Full) ↓
BabyAI	Full	88.97 ± 1.60	100.00 ± 0.00
	Random fixed	74.42 ± 5.21	51.02 ± 2.63
	Nodewise gate	70.22 ± 7.41	50.99 ± 2.09
	Global Fast/Slow	70.64 ± 5.24	51.54 ± 1.97
	Ours	$\mathbf{87.42 \pm 2.83}$	49.12 ± 3.09
MiniHack	Full	89.10 ± 1.59	100.00 ± 0.00
	Random fixed	55.12 ± 5.54	48.25 ± 4.33
	Nodewise gate	86.52 ± 1.15	47.50 ± 3.57
	Global Fast/Slow	86.12 ± 3.35	49.87 ± 6.09
	Ours	$\mathbf{88.10 \pm 1.21}$	48.86 ± 4.28
Overcooked-AI	Full	95.11 ± 4.47	100.00 ± 0.00
	Random fixed	61.06 ± 14.01	21.13 ± 3.79
	Nodewise gate	75.33 ± 6.17	21.35 ± 4.02
	Global Fast/Slow	79.00 ± 7.78	21.30 ± 4.26
	Ours	$\mathbf{95.11 \pm 4.47}$	22.26 ± 4.63

Table 6: Paired benchmark-level success differences, Ours minus each comparator, with 95% percentile-bootstrap intervals. The six aligned model-seed–evaluation-bank slots are resampled as paired units. Within each slot, task or layout success is averaged before forming the benchmark-level difference. Values are percentage points.

Benchmark	Random fixed	Nodewise gate	Global Fast/Slow	Full	Flat-BC
BabyAI	$+13.00\,[+9.50, +16.25]$	$+17.19\,[+13.47, +20.67]$	$+16.78\,[+13.69, +19.97]$	$-1.56\,[-2.75, -0.39]$	$+35.81\,[+30.67, +41.00]$
MiniHack	$+32.98\,[+16.19, +50.88]$	$+1.58\,[-1.02, +4.50]$	$+1.98\,[-2.54, +8.40]$	$-1.00\,[-2.60, +0.04]$	$+17.46\,[+6.10, +32.06]$
Overcooked-AI	$+34.06\,[+25.67, +42.39]$	$+19.78\,[+12.28, +27.39]$	$+16.11\,[+5.00, +28.39]$	$+0.00\,[+0.00, +0.00]$	$+30.89\,[+19.06, +43.28]$

The `asymmetric_advantages` condition reuses the 54-node `cramped_room` scaffold and retrains its parameters for the new layout; `large_room` uses a separately generated 66-node scaffold. This comparison examines reuse of the recursive factorization across spatial contexts alongside adaptation to a larger coordination setting.

Table 7: Complete BabyAI results. Expanded nodes are averaged over primitive-action decisions, and Expansion is normalized by the corresponding Full recursive scaffold. Values are mean±SD over six independent training runs.

Task	Method	Success (%) ↑	Expanded nodes ↓	Expansion (% Full) ↓
GoToObj	Flat-BC	88.17 ± 10.78	–	–
	Random fixed	99.83 ± 0.41	11.67 ± 1.03	64.81 ± 5.74
	Nodewise gate	99.00 ± 0.89	11.43 ± 0.97	63.50 ± 5.41
	Global Fast/Slow	100.00 ± 0.00	11.43 ± 1.23	63.50 ± 6.85
	Ours	98.83 ± 1.60	11.45 ± 1.06	63.61 ± 5.86
	Full	98.83 ± 0.98	18.00 ± 0.00	100.00 ± 0.00
OpenDoor	Flat-BC	90.67 ± 4.32	–	–
	Random fixed	77.00 ± 4.82	7.67 ± 1.03	38.33 ± 5.16
	Nodewise gate	80.67 ± 7.17	7.66 ± 0.85	38.30 ± 4.26
	Global Fast/Slow	89.67 ± 3.39	7.62 ± 0.86	38.10 ± 4.28
	Ours	84.83 ± 5.98	7.51 ± 1.05	37.55 ± 5.26
	Full	90.17 ± 2.32	20.00 ± 0.00	100.00 ± 0.00
PickupLoc	Flat-BC	86.67 ± 2.42	–	–
	Random fixed	81.83 ± 3.71	15.67 ± 1.03	68.12 ± 4.49
	Nodewise gate	84.00 ± 3.90	16.02 ± 1.23	69.64 ± 5.34
	Global Fast/Slow	84.67 ± 1.37	15.37 ± 1.22	66.82 ± 5.32
	Ours	84.50 ± 3.51	15.84 ± 1.03	68.88 ± 4.47
	Full	83.00 ± 3.03	23.00 ± 0.00	100.00 ± 0.00
UnblockPickup	Flat-BC	38.83 ± 24.30	–	–
	Random fixed	73.00 ± 8.44	19.67 ± 1.86	57.84 ± 5.48
	Nodewise gate	65.00 ± 4.00	19.51 ± 1.04	57.38 ± 3.07
	Global Fast/Slow	51.83 ± 27.01	20.06 ± 0.87	59.00 ± 2.57
	Ours	85.67 ± 6.35	18.84 ± 1.50	55.40 ± 4.40
	Full	87.83 ± 6.05	34.00 ± 0.00	100.00 ± 0.00
UnlockPickup	Flat-BC	5.33 ± 8.29	–	–
	Random fixed	58.33 ± 19.33	21.33 ± 1.03	42.67 ± 2.07
	Nodewise gate	56.67 ± 17.49	21.14 ± 1.58	42.28 ± 3.16
	Global Fast/Slow	66.33 ± 18.85	21.55 ± 1.12	43.11 ± 2.24
	Ours	91.00 ± 2.97	17.01 ± 6.13	34.03 ± 12.27
	Full	89.17 ± 3.06	50.00 ± 0.00	100.00 ± 0.00
BlockedUnlockPickup	Flat-BC	0.00 ± 0.00	–	–
	Random fixed	56.50 ± 17.44	22.00 ± 1.55	34.38 ± 2.42
	Nodewise gate	36.00 ± 21.72	22.29 ± 1.89	34.82 ± 2.95
	Global Fast/Slow	31.33 ± 22.58	24.75 ± 0.85	38.68 ± 1.33
	Ours	79.67 ± 8.26	22.55 ± 1.49	35.24 ± 2.33
	Full	84.83 ± 2.79	64.00 ± 0.00	100.00 ± 0.00

Table 8: Complete MiniHack results under the coarse-subgoal protocol. Expanded nodes are averaged over primitive-action decisions, and Expansion is normalized by the corresponding Full recursive scaffold. Values are mean±SD over six independent training runs.

Task	Method	Success (%) ↑	Expanded nodes ↓	Expansion (% Full) ↓
Room-5x5	Flat-BC	85.17 ± 7.96	–	–
	Random fixed	92.33 ± 3.33	7.33 ± 0.52	61.11 ± 4.30
	Nodewise gate	94.83 ± 4.45	7.26 ± 1.33	60.47 ± 11.09
	Global Fast/Slow	100.00 ± 0.00	8.06 ± 1.45	67.21 ± 12.09
	Ours	95.17 ± 6.77	7.66 ± 0.50	63.79 ± 4.17
	Full	100.00 ± 0.00	12.00 ± 0.00	100.00 ± 0.00
Room-15x15	Flat-BC	9.67 ± 5.68	–	–
	Random fixed	46.67 ± 5.82	11.33 ± 1.03	94.44 ± 8.61
	Nodewise gate	47.17 ± 5.34	9.51 ± 3.77	79.26 ± 31.44
	Global Fast/Slow	47.33 ± 5.32	11.89 ± 0.09	99.06 ± 0.75
	Ours	47.00 ± 5.40	10.99 ± 1.10	91.60 ± 9.20
	Full	47.33 ± 5.32	12.00 ± 0.00	100.00 ± 0.00
Wear	Flat-BC	85.33 ± 8.59	–	–
	Random fixed	59.17 ± 18.23	8.00 ± 2.37	44.44 ± 13.15
	Nodewise gate	94.50 ± 5.09	8.98 ± 2.85	49.89 ± 15.85
	Global Fast/Slow	87.50 ± 19.67	7.60 ± 2.61	42.20 ± 14.52
	Ours	96.83 ± 4.58	8.11 ± 2.54	45.06 ± 14.12
	Full	96.67 ± 4.46	18.00 ± 0.00	100.00 ± 0.00
Eat-coarse	Flat-BC	96.00 ± 3.74	–	–
	Random fixed	72.67 ± 8.80	9.67 ± 1.86	53.70 ± 10.34
	Nodewise gate	93.67 ± 7.87	8.77 ± 2.90	48.74 ± 16.13
	Global Fast/Slow	81.83 ± 25.87	8.53 ± 1.72	47.39 ± 9.55
	Ours	98.67 ± 1.37	9.97 ± 2.21	55.41 ± 12.29
	Full	99.50 ± 0.84	18.00 ± 0.00	100.00 ± 0.00
Pray	Flat-BC	91.33 ± 2.16	–	–
	Random fixed	78.67 ± 14.28	14.67 ± 0.52	69.84 ± 2.46
	Nodewise gate	96.50 ± 4.68	15.01 ± 0.20	71.45 ± 0.95
	Global Fast/Slow	98.00 ± 3.16	15.39 ± 2.04	73.30 ± 9.70
	Ours	98.50 ± 1.64	14.66 ± 0.31	69.79 ± 1.49
	Full	98.67 ± 2.16	21.00 ± 0.00	100.00 ± 0.00
LavaCross-Ring	Flat-BC	75.17 ± 5.15	–	–
	Random fixed	37.83 ± 12.11	5.00 ± 0.89	27.78 ± 4.97
	Nodewise gate	87.33 ± 5.09	5.82 ± 0.69	32.34 ± 3.84
	Global Fast/Slow	89.00 ± 2.68	5.35 ± 0.62	29.72 ± 3.45
	Ours	84.17 ± 5.00	5.15 ± 0.94	28.60 ± 5.24
	Full	84.17 ± 5.00	18.00 ± 0.00	100.00 ± 0.00
LavaCross-Potion	Flat-BC	86.33 ± 7.42	–	–
	Random fixed	37.67 ± 15.11	4.33 ± 1.03	27.08 ± 6.45
	Nodewise gate	91.00 ± 3.46	4.44 ± 1.49	27.75 ± 9.30
	Global Fast/Slow	88.50 ± 2.74	5.23 ± 1.39	32.70 ± 8.70
	Ours	89.00 ± 2.68	4.68 ± 1.20	29.26 ± 7.52
	Full	88.50 ± 2.74	16.00 ± 0.00	100.00 ± 0.00
WoD-Hard	Flat-BC	36.17 ± 8.23	–	–
	Random fixed	16.00 ± 13.05	1.67 ± 0.52	7.58 ± 2.35
	Nodewise gate	87.17 ± 5.85	2.22 ± 0.20	10.08 ± 0.90
	Global Fast/Slow	96.83 ± 4.22	1.63 ± 0.58	7.40 ± 2.62
	Ours	95.50 ± 3.99	1.62 ± 0.40	7.37 ± 1.82
	Full	98.00 ± 2.10	22.00 ± 0.00	100.00 ± 0.00

Table 9: Complete Overcooked-AI results across three layouts. Success@2 is the percentage of rollouts delivering at least two soups within 100 primitive joint-action steps. Expanded nodes are averaged over player-0 decisions, and Expansion is normalized by the corresponding Full recursive scaffold. Values are mean±SD over six independent training runs.

Layout	Method	Success@2 (%) ↑	Expanded nodes ↓	Expansion (% Full) ↓
cramped_room	Flat-BC	63.50 ± 20.64	–	–
	Random fixed	81.67 ± 7.87	9.96 ± 3.47	18.44 ± 6.43
	Nodewise gate	84.33 ± 12.19	9.83 ± 3.43	18.21 ± 6.34
	Global Fast/Slow	73.83 ± 24.83	10.86 ± 2.42	20.11 ± 4.49
	Ours	96.67 ± 2.16	12.15 ± 1.04	22.51 ± 1.93
	Full	96.67 ± 2.16	54.00 ± 0.00	100.00 ± 0.00
asymmetric_advantages	Flat-BC	57.50 ± 44.03	–	–
	Random fixed	57.33 ± 17.94	12.00 ± 3.07	22.21 ± 5.68
	Nodewise gate	77.50 ± 20.22	12.18 ± 3.05	22.56 ± 5.64
	Global Fast/Slow	86.17 ± 19.93	11.46 ± 3.23	21.23 ± 5.98
	Ours	97.33 ± 3.08	11.88 ± 2.98	21.99 ± 5.53
	Full	97.33 ± 3.08	54.00 ± 0.00	100.00 ± 0.00
large_room	Flat-BC	71.67 ± 22.57	–	–
	Random fixed	44.17 ± 28.36	15.00 ± 8.68	22.73 ± 13.16
	Nodewise gate	64.17 ± 22.22	15.38 ± 9.38	23.30 ± 14.22
	Global Fast/Slow	77.00 ± 29.02	14.90 ± 8.64	22.58 ± 13.09
	Ours	91.33 ± 12.96	14.70 ± 8.55	22.28 ± 12.96
	Full	91.33 ± 12.96	66.00 ± 0.00	100.00 ± 0.00

D Representation, Supervision, and Inference Ablations

NoLeaf removes direct predicate-label supervision while retaining the scaffold, action objective, and contract–expand inference. Its leaf values are learned as latent action-support factors. **Flat predicate MLP** receives the same predicted predicates and decision context, replacing recursive AND/OR composition with a direct action head. **No-action** retains the scaffold and predicate supervision but removes action-level learning. **RF-order + Stop** uses the same trained recursive-energy checkpoint and scaffold as Ours. It replaces Ours' predicted-radius expansion priority with the fixed randomized frontier order of the Random fixed baseline, while retaining Ours' predicted margin-separation stopping condition after each expansion.

Table 10: Task-level success (%) for representation, supervision, and inference-allocation ablations across BabyAI, MiniHack, and Overcooked-AI. Values are mean±SD over six reported evaluations.

(a) BabyAI

Task	Ours	NoLeaf	Flat MLP	RF-order + Stop
`GoToObj`	98.83 ± 1.60	74.00 ± 18.51	98.67 ± 1.21	99.50 ± 0.84
`OpenDoor`	84.83 ± 5.98	89.83 ± 6.79	87.50 ± 4.23	88.33 ± 6.59
`PickupLoc`	84.50 ± 3.51	75.00 ± 4.98	79.50 ± 5.01	84.50 ± 2.88
`UnblockPickup`	85.67 ± 6.35	78.33 ± 8.36	20.83 ± 3.92	85.00 ± 8.90
`UnlockPickup`	91.00 ± 2.97	99.50 ± 0.84	72.83 ± 17.51	86.17 ± 6.85
`BlockedUnlockPickup`	79.67 ± 8.26	35.33 ± 12.88	43.83 ± 32.80	78.89 ± 7.78
Macro	$\mathbf{87.42 \pm 2.83}$	75.33 ± 6.33	67.19 ± 8.17	87.06 ± 2.29

(b) MiniHack

Task	Ours	NoLeaf	Flat MLP	RF-order + Stop
`Room-5x5`	95.17 ± 6.77	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
`Room-15x15`	47.00 ± 5.40	42.83 ± 5.88	44.67 ± 6.09	47.33 ± 5.32
`Wear`	96.83 ± 4.58	94.67 ± 7.50	66.67 ± 51.64	97.33 ± 3.39
`Eat-coarse`	98.67 ± 1.37	94.17 ± 6.08	33.33 ± 51.64	99.83 ± 0.41
`Pray`	98.50 ± 1.64	37.33 ± 42.94	98.67 ± 2.16	98.50 ± 1.64
`LavaCross-Ring`	84.17 ± 5.00	81.00 ± 0.00	98.33 ± 2.58	84.17 ± 5.00
`LavaCross-Potion`	89.00 ± 2.68	80.33 ± 10.42	98.17 ± 2.86	88.50 ± 2.74
`WoD-Hard`	95.50 ± 3.99	95.33 ± 4.59	98.00 ± 2.90	97.33 ± 2.88
Macro	$\mathbf{88.10 \pm 1.21}$	78.21 ± 5.64	79.73 ± 11.86	89.13 ± 1.59

(c) Overcooked-AI

Layout	Ours	NoLeaf	Flat MLP	RF-order + Stop
`cramped_room`	96.67 ± 2.16	95.83 ± 3.43	96.67 ± 2.80	95.67 ± 2.80
`asymmetric_advantages`	97.33 ± 3.08	92.67 ± 17.96	99.67 ± 0.82	97.50 ± 4.46
`large_room`	91.33 ± 12.96	57.67 ± 35.38	37.67 ± 20.03	92.83 ± 5.56
Macro	$\mathbf{95.11 \pm 4.47}$	82.06 ± 14.03	78.00 ± 6.72	95.33 ± 1.08

Table 10 shows that predicate labels are beneficial but are not required for NoLeaf to learn useful action-support factors. Its losses are task-dependent: the largest drops include `BlockedUnlockPickup`, MiniHack `Pray`, and Overcooked-AI `large_room`. In a separate run with 2,000 demonstrations, NoLeaf reaches $93.33 \pm 3.10\%$ success on `BlockedUnlockPickup`, compared with $35.33 \pm 12.88\%$ at the original training budget. This recovery is consistent with predicate labels providing a useful training signal that reduces the demonstrations needed on this task. Flat predicate MLP is competitive on several individual tasks, while recursive composition has larger advantages on BabyAI obstacle-clearing tasks, MiniHack `Wear` and `Eat-coarse`, and Overcooked-AI `large_room`.

RF-order + Stop also reaches success close to Ours on most tasks, showing that predicted margin-based stopping can recover much of the performance lost by budget-matched Random fixed. However, success alone does not capture the cost of reaching the stopping condition. We therefore compare their expansion rates in Table 11.

Table 11: Inference-allocation ablation across the three benchmark families. RF-order + Stop combines the fixed randomized expansion order of Random fixed with the predicted margin stopping rule of Ours. Success is mean±SD over six reported evaluations; expansion is the macro mean relative to Full. The Random fixed and Ours rows reproduce the main benchmark summaries.

Benchmark	Method	Success (%) ↑	Expansion (% Full) ↓
BabyAI	Random fixed	74.42 ± 5.21	51.02
	RF-order + Stop	87.06 ± 2.29	64.16
	Ours	87.42 ± 2.83	49.12
MiniHack	Random fixed	55.12 ± 5.54	48.25
	RF-order + Stop	89.13 ± 1.59	69.58
	Ours	88.10 ± 1.21	48.86
Overcooked-AI	Random fixed	61.06 ± 14.01	21.13
	RF-order + Stop	95.33 ± 1.08	49.10
	Ours	95.11 ± 4.47	22.26

Together, the ablations identify distinct sources of performance. NoLeaf shows that action supervision can learn useful latent factors without direct predicate labels, while the additional-demonstration result illustrates how predicate supervision can help at a smaller data budget. Flat predicate MLP shows that direct prediction can work well on individual tasks, whereas the recursive structure is especially valuable for decisions involving multiple prerequisites and interactions. In the reported No-action training setup, removing action-level learning while retaining the scaffold and predicate supervision yields 0% success across the evaluated tasks. This shows that the supplied tree and predicate targets do not suffice under that ablation's training setup.

The inference ablation further separates stopping from expansion selection. RF-order + Stop restores high success, but expands 64.16%, 69.58%, and 49.10% of Full on BabyAI, MiniHack, and Overcooked-AI. Ours achieves comparable benchmark-level success with 49.12%, 48.86%, and 22.26% expansion, respectively. Thus, predicted-radius-guided selection reaches the stopping condition with substantially fewer expanded nodes than randomized selection: the gain comes from choosing where to expand, in addition to knowing when to stop.

E Transfer to More Complex Task Contexts

The controller and error-radius head trained on BabyAI `UnblockPickup` are evaluated on retained `BossLevel` missions without OOD-specific retraining or recalibration. The retained missions use the training skill repertoire in generally more complex scenarios; missions requiring unseen skills, including unlocking locked doors and `PutNext`, are excluded.

Figure 5 reports closed-loop performance. Under transfer, the controller shifts toward finer evaluation and maintains success close to Full. This is consistent with predicted margin-based inference expanding more subtrees when contracted energies provide less separation between candidate actions.

Table 12: Reliability of learned error radii and early stopping under OOD evaluation. Values are mean±SD over six independent runs. Residual-margin failure is the complement of residual-margin validity among early-stopped decisions; it is based on the propagated residuals $R_a^{S_a}$ and is distinct from the root-undercoverage severity event $\rho \geq 1$.

Metric	OOD result
Early stopping (%)	34.83 ± 16.01
Residual-margin validity (%)	$\mathbf{99.62 \pm 0.93}$
Realized-margin validity (%)	$\mathbf{100.00 \pm 0.00}$
Early-stop Full-winner disagreement (%)	$\mathbf{0.00 \pm 0.00}$
Residual-margin failure (%)	0.38 ± 0.93

The decision-level audit uses the definitions in Section 4.4. Residual-margin failure is the complement of residual-margin validity among early-stopped decisions; it is based on propagated R and differs

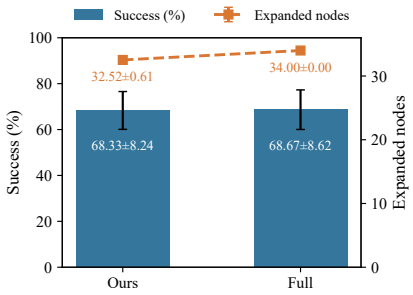


Figure 5: Closed-loop transfer to retained `BossLevel` missions. Success is $68.33 \pm 8.24\%$ for Ours and $68.67 \pm 8.62\%$ for Full; Ours expands 32.52 ± 0.61 of 34 nodes. Values are mean±sample SD across six runs with 100 episodes per run. The trained error-radius predictor is used without OOD-specific recalibration.

from the realized severity event $\rho \geq 1$. Table 12 shows that the remaining early stops retain high margin validity and no observed Full-winner changes.

The severity distribution is concentrated entirely in the mild range: $51.13 \pm 22.48\%$ of early-stopped decisions have $\rho = 0$, and $48.87 \pm 22.48\%$ have $0 < \rho \leq 0.25$. No early-stopped decisions have $\rho > 0.25$, and no severity bin contains a Full-winner change. Thus, the learned controller combines greater expansion under this shift with reliable early stopping on the decisions selected by the predicted rule.

F Additional Qualitative and Diagnostic Results

F.1 Severity of Predicted Early Stops

For an early-stopped decision, define realized root undercoverage and its severity by

$$u_a = (d_a - B_a^{S_a})_+, \qquad \rho = \max_{a \neq a^\star} \frac{u_a + u_{a^\star}}{\gamma_a}. \tag{15}$$

At $\rho = 0$, the predicted root radii cover the observed deviations. The range $0 < \rho < 1$ permits undercoverage while retaining strict winner separation: $d_a + d_{a^\star} \leq B_a^{S_a} + B_{a^\star}^{S_{a^\star}} + u_a + u_{a^\star} < F_a^{S_a} - F_{a^\star}^{S_{a^\star}}$. The boundary $\rho = 1$ can permit a Full tie.

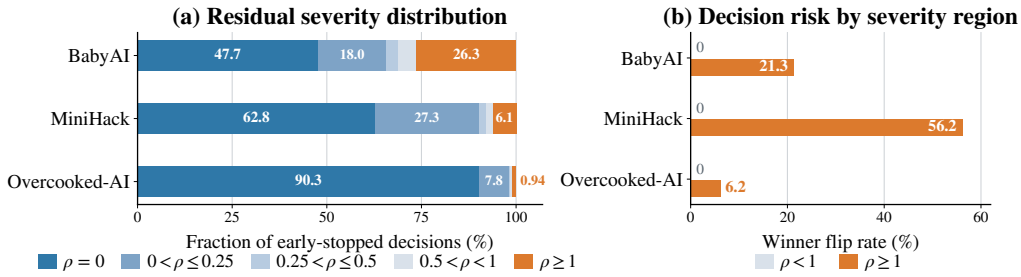


Figure 6: **Undercoverage relative to the remaining action margin.** Left: decision-level frequencies among early-stopped decisions, with a different weighting from Table 1. Right: Full-winner changes within the indicated severity regions. The $\rho < 1$ group has no observed winner changes; no audited sample has $\rho = 1$.

Figure 6 shows that most audited early stops lie below the winner-preservation boundary. The $\rho \geq 1$ tail marks decisions for which the post-hoc sufficient condition no longer guarantees the Full winner; it is not the observed flip rate. The right panel reports the actual winner changes. Although some decisions do flip, the aggregate success gap to Full remains small on BabyAI and MiniHack and is zero on Overcooked-AI.

F.2 Local Resolution and Learned Contraction

Figure 2 combines a BabyAI rollout with three highlighted decision states. The mission is to pick up the green key. The high-level interface supplies successive `CrossDoor` and `Pickup` subgoals, while the low-level controller chooses primitive actions from the active subgoal and observation context. Trajectory colors identify the subgoal stages, and arrow thickness indicates the amount of recursive expansion at each decision.

All three highlighted states select `forward`, but use different evaluation resolutions. In the contracted case, a macro-energy summarizes the action tree and the hollow descendants remain implicit. In the expanded case, recursive evaluation exposes the supporting factors. The intermediate case retains blue contracted subtrees alongside red expanded branches, showing both resolutions coexisting within one action evaluation. Thus, the extent and location of explicit expansion vary with the decision context, even when the selected primitive action is unchanged.

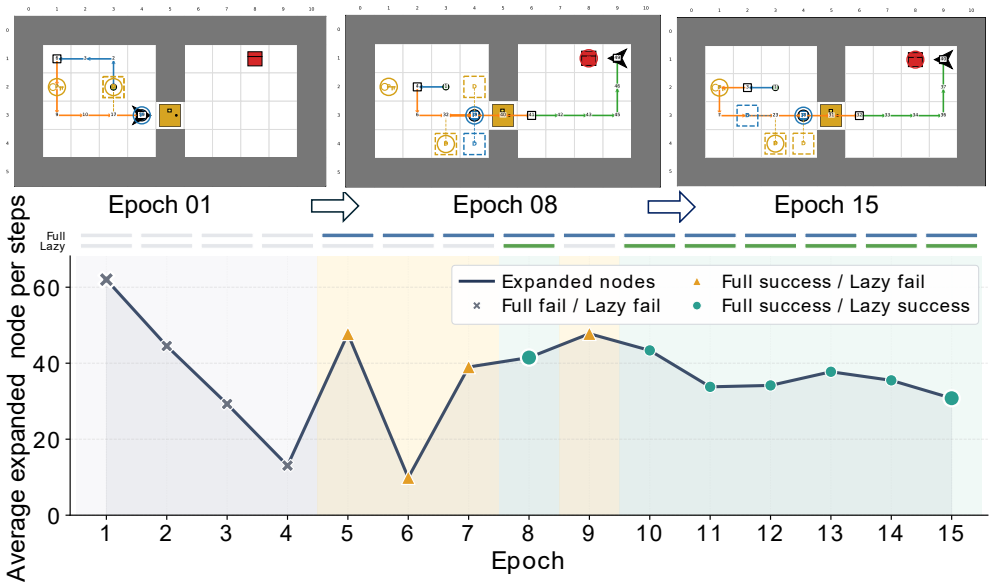


Figure 7: Learning progression on the same evaluation case across training epochs. The agent gradually learns a key-and-obstacle routine: pick up the key, temporarily drop it to clear an obstacle, recover the key, unlock/open the door, discard the key after use, cross into the next room, and pick up the target object. At later epochs, lazy contract–expand inference succeeds with fewer expanded nodes and fewer environment steps, indicating that expanded reasoning has been compressed into reliable contracted evaluations.

The learning-progression example links decreasing expansion to the reuse of learned subtree summaries at comparable stages of the task.

F.3 State-Dependent Recursive Expansion

The MiniHack and Overcooked-AI examples contrast routine decisions with interactions requiring more recursive evaluation. Command choices, narrow passages, shared resources, and immediate coordination can activate deeper evaluation, while familiar movement and isolated interactions often retain compressed subtrees. These are qualitative illustrations of the allocation behavior rather than additional task-performance metrics.

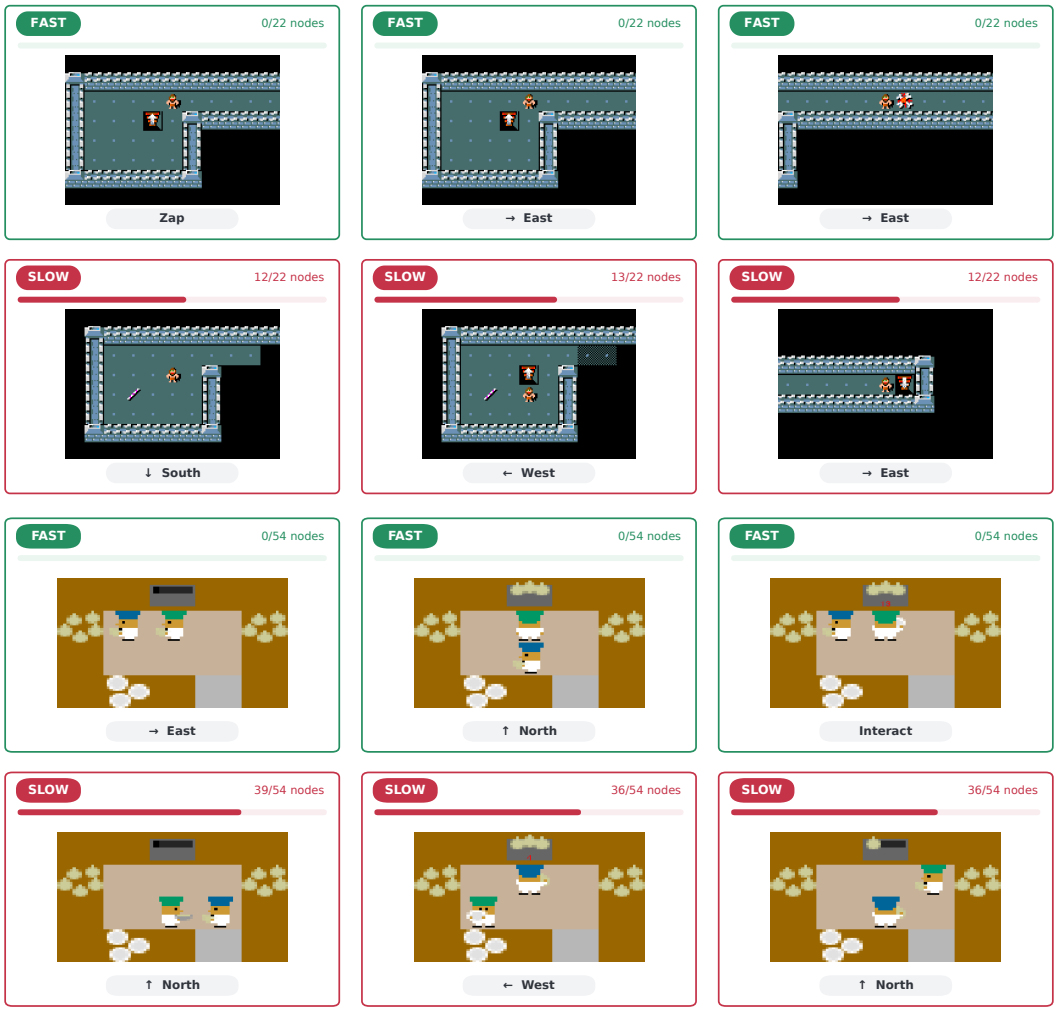


Figure 8: Representative low- and high-expansion decisions. **Top: MiniHack.** Fast states are resolved without explicit recursive expansion, whereas selected ambiguous states require substantially deeper evaluation of the 22-node scaffold. **Bottom: Overcooked-AI.** The same controller can either stop early for a decision without entering the recursive scaffold or expand a large fraction of the 54-node structure when competing alternatives remain unresolved. These examples illustrate that explicit expansion varies with the current decision context rather than fixed by the primitive action or task stage alone.

F.4 Training Dynamics

The curves report wake and sleep losses, explicit expansion relative to Full, and local undercoverage rate over six independent training runs. Violation rate is one minus coverage: the fraction of checked learned error radii whose realized contraction gaps exceed their predictions. These diagnostics use expert-demonstration contexts in the held-out 20% partition of the 80/20 training–validation split.

Losses stabilize while contraction increases to task-dependent levels. The different final expansion levels reflect adaptation to the learned task structure. Local validation coverage complements the closed-loop, margin-aware decision audit in Section 4.4.

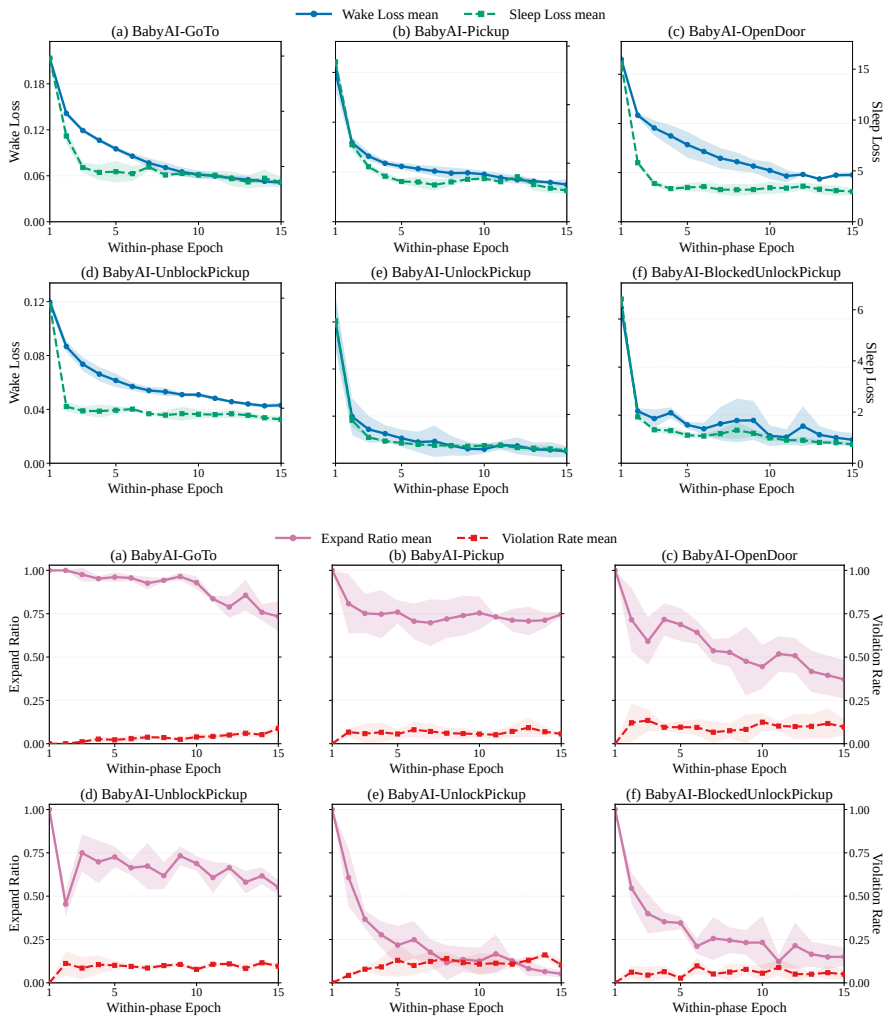


Figure 9: Training dynamics on the six BabyAI tasks. **Top:** wake and sleep losses over the wake–sleep training phase. **Bottom:** expansion ratio and local undercoverage rate. The violation rate is computed as one minus validation coverage and measures the fraction of learned error radii that fail to cover their realized contraction gaps. Optimization losses generally decrease and stabilize, while explicit recursive expansion is progressively reduced to task-dependent levels with low local undercoverage rates. Curves show the mean over six independent training runs, and shaded regions indicate ± 1 standard deviation.

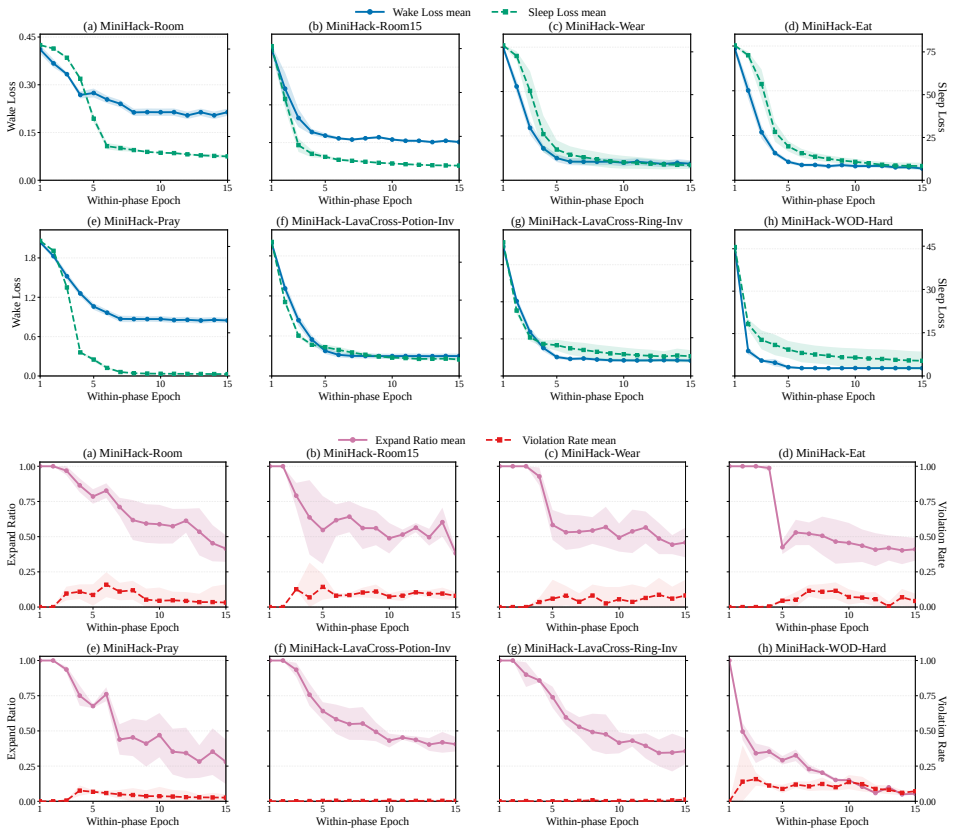


Figure 10: Training dynamics on the eight MiniHack tasks. **Top:** wake and sleep losses over the wake–sleep training phase. **Bottom:** expansion ratio and local undercoverage rate. Wake and sleep objectives decrease substantially during early training and subsequently stabilize. Meanwhile, the controller learns substantially different contraction levels across tasks while maintaining reliable learned error radii at different expansion levels. Curves show the mean over six independent training runs, and shaded regions indicate ± 1 standard deviation.

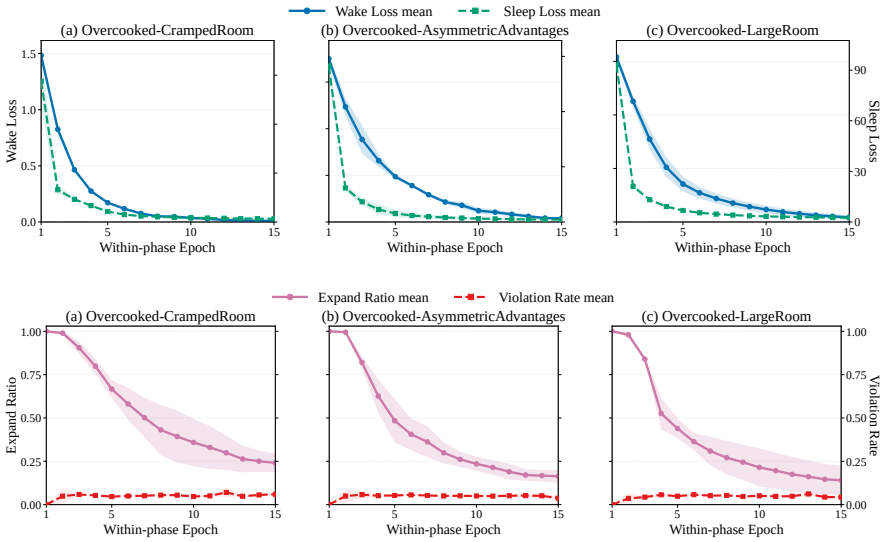


Figure 11: Training dynamics on the three Overcooked-AI layouts. **Top:** wake and sleep losses over the wake–sleep training phase. **Bottom:** expansion ratio and local undercoverage rate. The controller transitions from predominantly expanded recursive evaluation early in training toward substantially more contracted evaluation, while local undercoverage rates remain low across the three coordination settings. Curves show the mean over six independent training runs, and shaded regions indicate ± 1 standard deviation.

F.5 Contraction–Success Sweep

For `PickupLoc-v0`, we freeze the main-result checkpoints and impose a uniform per-decision expansion budget $b \in \{0, 3, 6, 9, 12, 14, 17, 20, 23\}$ by setting the minimum and maximum expansion counts to the same value. Every setting uses the same six 100-environment evaluation banks as the main results.

The sweep shows rising success and shorter episodes as recursive evaluation increases, followed by diminishing gains near Full. The natural adaptive controller lies above neighboring uniform-budget settings, supporting the value of deciding where to refine rather than using the same expansion count at every state. Its success remains close to Full while using substantially fewer expanded nodes.

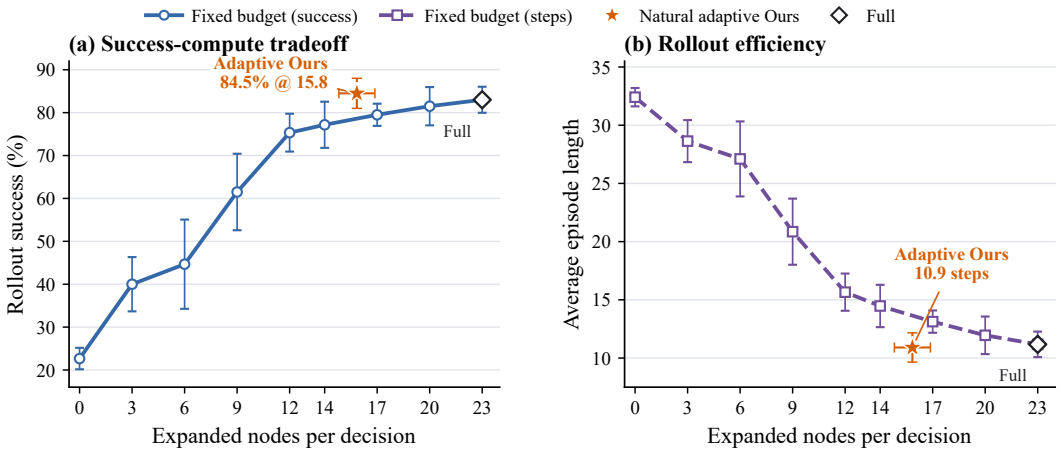


Figure 12: Contraction–success trade-off on `PickupLoc-v0`. The blue solid curve shows success and the purple dashed curve shows episode length under a uniform expansion budget. The orange star is the natural state-adaptive controller used in the main table, and the diamond denotes Full evaluation. All settings use the same retained checkpoints and the same 100-environment evaluations. Error bars show mean ± sample standard deviation across the six evaluation runs. Increasing recursive evaluation raises success and shortens episodes, while adaptive allocation approaches Full performance with substantially fewer expanded nodes.

G Detailed Expansion and Decision Latency Analysis

We complement the closed-loop rollout results with fixed-state expansion and decision-latency benchmarks on all six BabyAI tasks, eight MiniHack tasks, and three Overcooked-AI layouts. For each task or layout, all methods evaluate identical decision states drawn from held-out validation expert trajectories. Observation histories, task contexts, structured states, and any precomputed action-conditioned consequence features are held fixed across methods. This protocol isolates low-level action-selection cost from differences in state visitation, trajectory length, environment stepping, and high-level planning. Ours and Full use the same trained checkpoint and differ only in recursive expansion at inference.

Each method is evaluated with batch size one. After warm-up, we time repeated evaluations of the fixed states, compute the median latency over states for each benchmark execution, and independently repeat the execution three times. P50 latency is reported as the mean±SD of these run-level medians. Expanded nodes count explicit recursive refinement, and Full agreement is the percentage of fixed states on which a method selects the same action as Full. All three benchmark families are timed on the same NVIDIA RTX 5080 Laptop GPU under the fixed-state, batch-size-one protocol described above. The timing hardware is separate from the training hardware reported in Appendix B.11. Because these measurements use held-out expert states rather than each policy's closed-loop state distribution, their expanded-node counts need not match the rollout averages in Tables 7–9. Latency comparisons here are paired within the fixed-state protocol and do not estimate whole-episode wall-clock time.

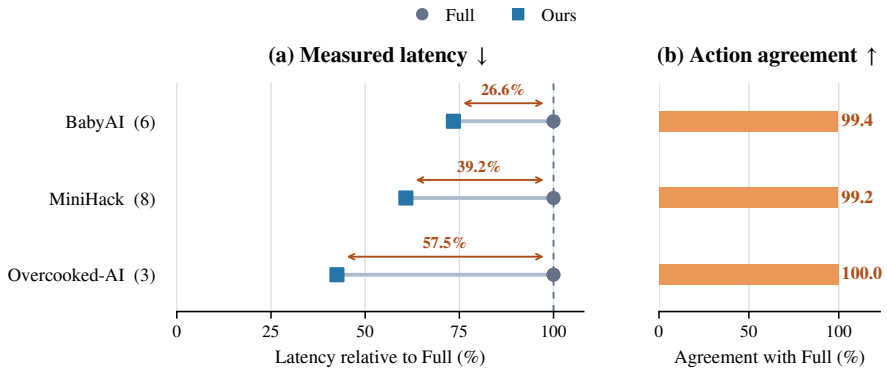


Figure 13: **Ours versus Full: decision latency and action agreement.** Within each task or layout, methods evaluate identical held-out expert decision states. The left panel shows the macro mean P50 latency of Ours relative to Full within each benchmark; arrows indicate the relative latency reduction. The right panel shows macro mean task-level action agreement with Full. Tables 13– 15 report per-task P50 latency mean±SD over three timing runs.

Table 13: Fixed-state expansion and decision latency on six BabyAI tasks. All methods receive identical held-out decision states within each task.

Task	Method	Expanded nodes ↓	P50 latency (ms) ↓	Full agreement (%) ↑
GoToObj	**Ours**	10.96	13.44 ± 6.17	100.0
	Full	18.00	17.05 ± 5.90	100.0
	Random fixed	11.00	13.27 ± 4.75	86.2
	Nodewise gate	10.41	12.33 ± 6.19	95.0
	Global Fast/Slow	9.68	10.97 ± 8.26	100.0
OpenDoor	**Ours**	8.20	6.75 ± 4.33	100.0
	Full	20.00	12.22 ± 2.38	100.0
	Random fixed	8.00	7.32 ± 1.88	97.5
	Nodewise gate	8.66	7.58 ± 3.64	100.0
	Global Fast/Slow	7.25	6.14 ± 5.52	100.0
PickupLoc	**Ours**	18.00	12.84 ± 3.50	100.0
	Full	23.00	13.63 ± 1.96	100.0
	Random fixed	15.00	10.78 ± 1.33	92.5
	Nodewise gate	15.16	10.80 ± 3.43	100.0
	Global Fast/Slow	15.53	10.04 ± 5.50	100.0
UnblockPickup	**Ours**	18.85	15.26 ± 1.00	97.5
	Full	34.00	20.32 ± 1.55	100.0
	Random fixed	19.00	15.29 ± 1.21	86.5
	Nodewise gate	19.60	15.54 ± 1.39	95.5
	Global Fast/Slow	19.21	18.74 ± 0.58	96.0
UnlockPickup	**Ours**	17.45	25.46 ± 2.54	100.0
	Full	50.00	41.18 ± 5.52	100.0
	Random fixed	22.00	32.23 ± 4.43	97.5
	Nodewise gate	19.14	28.73 ± 4.88	100.0
	Global Fast/Slow	21.25	9.43 ± 2.37	100.0
BlockedUnlockPickup	**Ours**	38.59	43.99 ± 1.83	98.8
	Full	64.00	55.96 ± 2.36	100.0
	Random fixed	38.00	47.39 ± 1.43	98.8
	Nodewise gate	25.06	42.89 ± 2.46	98.8
	Global Fast/Slow	36.80	51.69 ± 0.79	98.8

Table 14: Fixed-state expansion and wall-clock evaluation on eight MiniHack tasks.

Task (states)	Method	Expanded nodes ↓	P50 latency (ms) ↓	Full agreement (%) ↑
Room-5x5	**Ours**	8.310	23.340 ± 0.806	94.1
	Full	12.000	26.999 ± 1.487	100.0
	Random fixed	8.000	22.977 ± 1.072	75.4
	Nodewise gate	7.905	24.978 ± 0.701	96.9
	Global Fast/Slow	8.581	26.686 ± 0.808	100.0
Room-15x15	**Ours**	8.343	24.671 ± 1.077	100.0
	Full	12.000	29.153 ± 1.569	100.0
	Random fixed	8.000	24.730 ± 1.330	94.7
	Nodewise gate	8.813	28.407 ± 0.928	99.7
	Global Fast/Slow	9.212	29.123 ± 1.156	100.0
Wear	**Ours**	5.847	24.496 ± 1.519	100.0
	Full	18.000	36.781 ± 2.413	100.0
	Random fixed	6.000	20.835 ± 1.144	75.5
	Nodewise gate	5.867	23.329 ± 1.752	98.0
	Global Fast/Slow	6.000	7.077 ± 0.560	98.4
Eat-coarse	**Ours**	5.825	27.345 ± 1.214	100.0
	Full	18.000	37.433 ± 1.253	100.0
	Random fixed	6.000	20.914 ± 1.367	74.8
	Nodewise gate	5.293	22.140 ± 0.913	98.8
	Global Fast/Slow	5.561	6.710 ± 0.745	99.6
Pray	**Ours**	8.066	13.543 ± 0.689	100.0
	Full	21.000	21.132 ± 0.968	100.0
	Random fixed	8.000	14.890 ± 0.627	82.9
	Nodewise gate	8.425	16.131 ± 1.242	100.0
	Global Fast/Slow	7.473	3.608 ± 0.591	98.0
LavaCross-Ring	**Ours**	3.576	21.700 ± 3.013	100.0
	Full	18.000	39.808 ± 5.133	100.0
	Random fixed	4.000	18.135 ± 1.752	96.5
	Nodewise gate	3.635	18.302 ± 2.469	100.0
	Global Fast/Slow	3.388	6.224 ± 0.703	100.0
LavaCross-Potion	**Ours**	3.012	18.441 ± 2.689	100.0
	Full	16.000	36.687 ± 6.439	100.0
	Random fixed	3.000	15.496 ± 3.160	88.9
	Nodewise gate	3.272	17.909 ± 3.395	98.8
	Global Fast/Slow	2.963	5.755 ± 1.559	100.0
WoD-Hard	**Ours**	1.180	9.178 ± 1.343	99.3
	Full	22.000	39.624 ± 6.172	100.0
	Random fixed	1.000	10.163 ± 1.315	96.0
	Nodewise gate	0.710	10.393 ± 1.651	98.0
	Global Fast/Slow	0.513	4.022 ± 0.522	99.3

Table 15: Fixed-state expansion and wall-clock evaluation on three Overcooked-AI layouts.

Layout (seed)	Method	Expanded nodes ↓	P50 latency (ms) ↓	Full agreement (%) ↑
cramped_room	**Ours**	5.15	0.975 ± 0.042	100.0
	Full	54.00	3.313 ± 0.106	100.0
	Random fixed	9.15	1.799 ± 0.039	100.0
	Nodewise gate	8.85	1.765 ± 0.031	95.0
	Global Fast/Slow	10.80	0.681 ± 0.015	95.0
asymmetric_advantages	**Ours**	9.00	1.160 ± 0.053	100.0
	Full	54.00	3.261 ± 0.067	100.0
	Random fixed	10.60	1.861 ± 0.079	75.0
	Nodewise gate	8.35	1.628 ± 0.027	90.0
	Global Fast/Slow	10.80	0.666 ± 0.007	95.0
large_room	**Ours**	44.80	2.235 ± 0.131	100.0
	Full	66.00	3.708 ± 0.258	100.0
	Random fixed	45.55	4.668 ± 0.259	100.0
	Nodewise gate	45.00	4.435 ± 0.293	100.0
	Global Fast/Slow	39.60	4.149 ± 0.215	100.0

Across the three benchmarks, Ours has lower P50 decision latency than Full on all six BabyAI tasks, all eight MiniHack tasks, and all three Overcooked-AI layouts. Its median speedup on MiniHack ranges from $1.16\times$ to $4.31\times$, including a $1.56\times$ speedup on `Pray`; the three Overcooked-AI layouts show $3.40\times$, $2.82\times$, and $1.66\times$ speedups, respectively. Full action agreement ranges from 97.5% to 100.0% on BabyAI and from 94.1% to 100.0% on MiniHack, and reaches 100.0% on all three Overcooked-AI layouts. These fixed-state measurements show that reduced recursive expansion translates into lower measured median decision latency across every evaluated task and layout while retaining high agreement with Full.

H Additional Model and Training Details

The recursive energies, action policy, error-radius propagation, and inference procedure are defined in Sections 2 and 3. This appendix specifies the additional input and training conventions.

Leaf inputs and optional consequence features. With action and leaf embeddings e_a, e_ℓ, the predicate predictor is

$$s_\ell = \sigma(f_\theta(z_t, u_t^a, e_\ell, e_a)), \qquad \epsilon_\ell = -\log s_\ell.$$

Here $z_t = q_\phi(x_t)$ encodes the shared decision context. When available, an action-conditioned consequence representation is derived separately for each candidate action:

$$u_t^a = \mathrm{Agg}\big(\xi_{t,1}^a, \ldots, \xi_{t,M}^a\big), \qquad \xi_{t,m}^a \sim p_\omega(\xi \mid x_t, a).$$

The consequence generator may operate directly on x_t or internally on an encoding such as z_t. Thus, x_t contains information shared across candidate actions, whereas u_t^a represents optional action-specific predicted consequences derived from that shared context. A learned transition model, simulator, or programmatic transition function can supply these features. They augment the inputs to the action evaluator but are not part of the contract–expand mechanism itself.

Wake objective and replay. The complete wake objective is

$$\mathcal{L}_{\mathrm{wake}} = \mathcal{L}_{\mathrm{act}} + \sum_{j \in \mathcal{J}_{\mathrm{aux}}} \lambda_j \mathcal{L}_j.$$

Available auxiliary terms include fully expanded action supervision, predicate grounding, and transition learning when the consequence generator is jointly learned. After a wake update, replay stores $(x_t, a_t^E, \{S_{a,t}\}_{a \in \mathcal{A}})$. During sleep, the model recomputes the shared context encoding and any available action-conditioned consequence features, then evaluates expanded subtree targets for the replayed nodes. The macro regression uses stop-gradient targets; the asymmetric error-radius fitting loss is given in Eq. (7). Environment-specific objectives and hyperparameters appear in Appendix B.

I Deployment-Time Inference Procedure

Algorithm 1 Deployment-Time Margin-Aware Contract–Expand Inference

Require: shared decision context x_t, optional derived features $\{u_t^a\}_{a\in\mathcal{A}}$, action trees $\{T_a\}_{a\in\mathcal{A}}$, expansion budget K

Encode context $z_t = q_\phi(x_t)$
Initialize $S_a \leftarrow \varnothing$ for all $a \in \mathcal{A}$
Compute initial $F_a^{S_a}(x_t)$ and $B_a^{S_a}$ for all actions
for $k = 1, \dots, K$ **do**
  Let $a^\star \leftarrow \arg\min_a F_a^{S_a}(x_t)$
  if $a^\star$ satisfies Eq. (9) **then**
    break
  end if
  Form $\mathcal{C}$ and $\mathcal{Q}$ using Eqs. (10)–(11)
  if $\mathcal{Q} = \varnothing$ **then**
    break
  end if
  Select $(a, n) \leftarrow \arg\max_{(a,n)\in\mathcal{Q}} \text{priority}_a(n)$
  Update $S_a \leftarrow S_a \cup \{n\}$ and evaluate newly exposed children
  Update the affected action energies and propagated predicted radii
end for
Recompute final action scores for the final expansion sets
return $\hat{a}_t = \arg\min_a F_a^{S_a}(x_t)$

J Error-Radius Bounds and Winner-Preservation Proofs

Use the energies and predicted propagation from the main method. For an unresolved node, $q_n = (\Delta_n - \hat{r}_n)_+$. The residual $R_n^{S_a}$ uses Eq. (8) with q_n replacing $\hat{r}_n$: it is zero at leaves, equals q_n at unresolved internal nodes, sums at expanded AND nodes, and takes the maximum at expanded OR nodes. Write $R_a^{S_a} = R_{r_a}^{S_a}$. These realized residuals are used in post-hoc analysis; deployment uses the predicted B values.

Proposition 2 (Residual-aware error propagation)**.** *Fix an action a and a frontier-consistent expansion set S_a. Then*

$$\left|E_n^{\exp} - E_n^{S_a}\right| \le B_n^{S_a} + R_n^{S_a}$$

for every node n in the partial tree. In particular,

$$\left|F_a^{\exp}(x_t) - F_a^{S_a}(x_t)\right| \le B_a^{S_a} + R_a^{S_a}.$$

If all unresolved predicted error radii cover their realized contraction gaps, then $R_a^{S_a} = 0$ and the strict-coverage bound

$$\left|F_a^{\exp}(x_t) - F_a^{S_a}(x_t)\right| \le B_a^{S_a}$$

is recovered as a special case.

Corollary 1 (Margin-aware winner preservation)**.** *Suppose the deployment-time stopping rule returns $a^\star$. If*

$$R_a^{S_a} + R_{a^\star}^{S_{a^\star}} < \gamma_a, \qquad \forall a \neq a^\star,$$

then

$$F_{a^\star}^{\exp}(x_t) < F_a^{\exp}(x_t), \qquad \forall a \neq a^\star.$$

Therefore, $a^\star$ is also the winner under fully expanded evaluation.

Proof of Proposition 2. We proceed by induction over the subtree rooted at n.

For a leaf, $E_n^{S_a} = E_n^{\exp}$, so

$$|E_n^{\exp} - E_n^{S_a}| = 0 = B_n^{S_a} + R_n^{S_a}.$$

For an unresolved contracted internal node, $E_n^{S_a} = E_n^{\text{con}}$. By definition,

$$q_n = \left(|E_n^{\exp} - E_n^{\text{con}}| - \hat{r}_n\right)_+,$$

and therefore

$$|E_n^{\exp} - E_n^{S_a}| \le \hat{r}_n + q_n = B_n^{S_a} + R_n^{S_a}.$$

Now consider an expanded internal node. For an AND node, using the additive composition rule, the triangle inequality, and the induction hypothesis,

$$\begin{aligned}\left|E_n^{\exp} - E_n^{S_a}\right| &\le \sum_{c\in \mathrm{ch}(n)} \left|E_c^{\exp} - E_c^{S_a}\right| \\ &\le \sum_{c\in \mathrm{ch}(n)} \left(B_c^{S_a} + R_c^{S_a}\right) \\ &= B_n^{S_a} + R_n^{S_a}.\end{aligned}$$

For an OR node, define

$$A_\vee(x) = -\tau \log \sum_i \exp(-x_i/\tau).$$

Its gradient has nonnegative components that sum to one, so $A_\vee$ is 1-Lipschitz with respect to the ℓ_∞ norm. Hence

$$\begin{aligned}\left|E_n^{\exp} - E_n^{S_a}\right| &\le \max_{c\in \mathrm{ch}(n)} \left|E_c^{\exp} - E_c^{S_a}\right| \\ &\le \max_c \left(B_c^{S_a} + R_c^{S_a}\right) \\ &\le \max_c B_c^{S_a} + \max_c R_c^{S_a} \\ &= B_n^{S_a} + R_n^{S_a}.\end{aligned}$$

The result follows by induction. □

Proof of Corollary 1. By Proposition 2, for every competitor $a \neq a^\star$,

$$F_a^{\exp} \ge F_a^{S_a} - B_a^{S_a} - R_a^{S_a},$$

while

$$F_{a^\star}^{\exp} \le F_{a^\star}^{S_{a^\star}} + B_{a^\star}^{S_{a^\star}} + R_{a^\star}^{S_{a^\star}}.$$

Therefore,

$$\begin{aligned}F_a^{\exp} - F_{a^\star}^{\exp} &\ge F_a^{S_a} - F_{a^\star}^{S_{a^\star}} - B_a^{S_a} - B_{a^\star}^{S_{a^\star}} - R_a^{S_a} - R_{a^\star}^{S_{a^\star}} \\ &= \gamma_a - R_a^{S_a} - R_{a^\star}^{S_{a^\star}}.\end{aligned}$$

By assumption,

$$R_a^{S_a} + R_{a^\star}^{S_{a^\star}} < \gamma_a,$$

so

$$F_a^{\exp} > F_{a^\star}^{\exp}$$

for every $a \neq a^\star$. Thus $a^\star$ is also the fully expanded winner. □

K Example Recursive Energy Scaffold

The main BabyAI scaffolds are generated from task semantics and demonstrations by an LLM, manually audited, and then fixed before training. The `UnblockPickup` example combines target search, navigation, temporary obstacle clearing, safe dropping, doorway traversal, and final pickup. The table identifies action-specific semantic branches; the figures show their overall organization and an expanded obstacle-clearing subtree. Learned energies assign action preferences to this fixed factorization using the training and inference procedures in the main text.

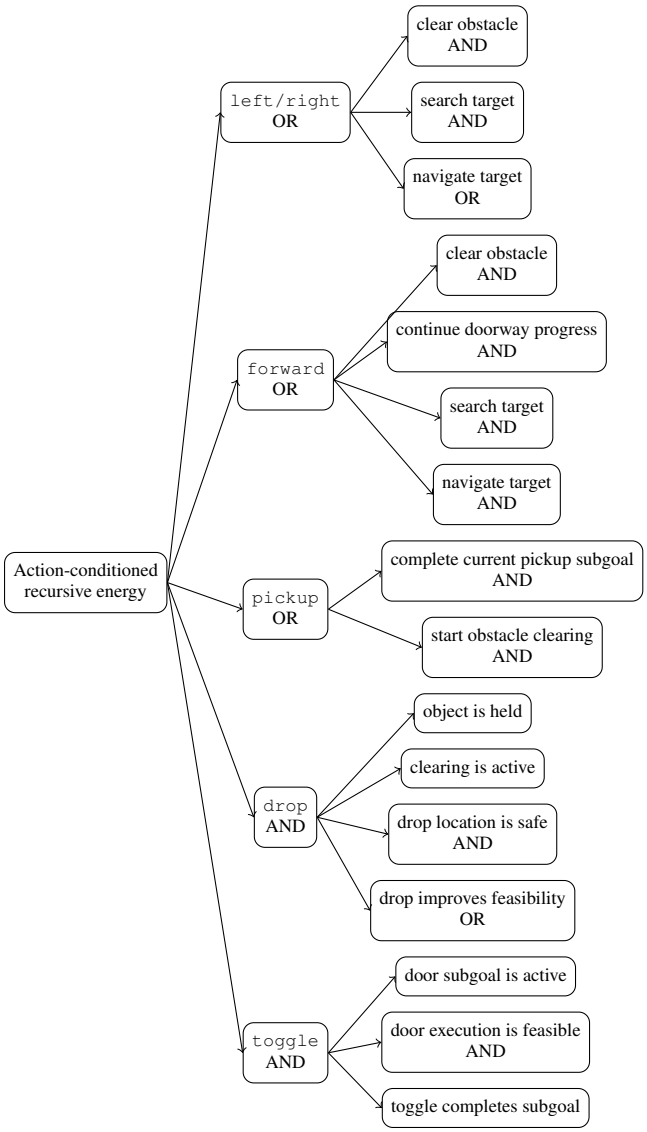


Figure 14: Compressed view of an action-conditioned scaffold. Each primitive action has a root with reusable task components; expandable branches expose more detailed factors.

Table 16: High-level semantic structure of one representative action-conditioned recursive energy scaffold.

Action	Root node	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

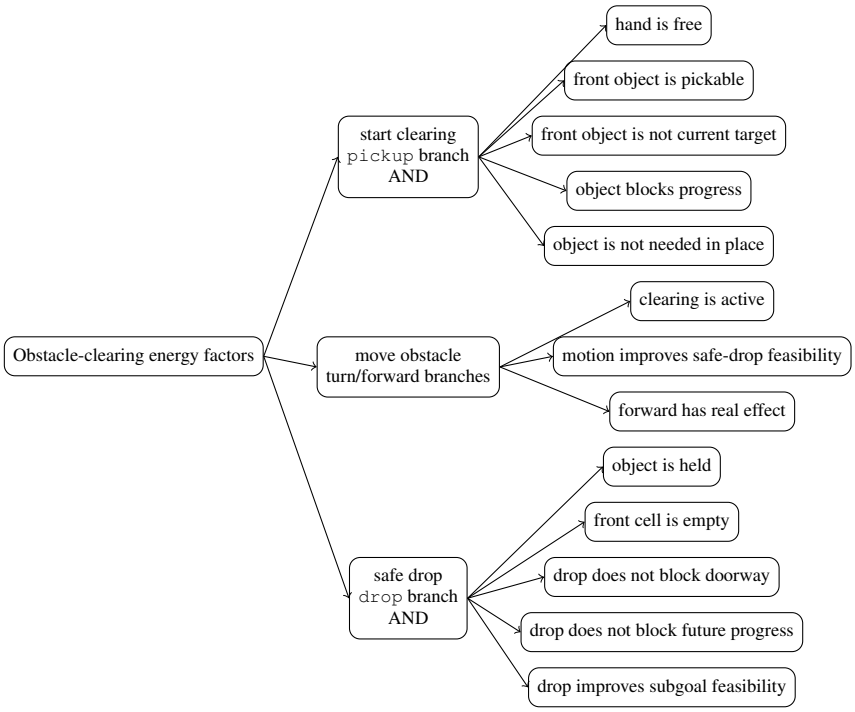


Figure 15: Expanded obstacle-clearing subtree, exposing guarded pickup, movement, and dropping conditions. AND and OR nodes encode prerequisites and alternatives.

L Automatic Scaffold Construction and Robustness

The recursive energy framework accepts scaffolds from different sources, including native task structure when available or structures induced from task descriptions and demonstrations. We instantiate the latter route with GPT-5.4 at temperature 0.2 and an executable verifier. The experiments in this appendix repeat BabyAI scaffold construction independently, then train and evaluate one controller per generated structure. The main benchmark experiments instead reuse a fixed scaffold across training seeds. The Boolean verifier guides structural construction; learned continuous energies supply downstream action preferences.

Induction. Tree-building demonstrations retain expert actions. Covering demonstrations have a labeler view with the current expert action withheld and a verifier view retaining it for evaluation. A predicate-library agent proposes Boolean facts about visible state, subgoals, history, inventory, affordances, and interaction context. A tree-builder agent composes these predicates into action-conditioned AND/OR trees for `left`, `right`, `forward`, `pickup`, `drop`, and `toggle`. The tree is normalized and schema-repaired before iterative verification.

Labeling and verification. At each round, an offline labeler annotates predicates on covering episodes, using the corresponding local observation, subgoal, inventory, and bounded history. Episodes are supplied together for annotation efficiency. The annotations serve construction and optional training supervision; deployed leaf values are predicted from runtime context. The verifier uses leaf energy 0 for true predicates and 100 otherwise, sums AND children, and takes the minimum over OR children. Its candidate set and agreement are

$$\mathcal{A}_{\min}(o_t) = \arg\min_a E_a(o_t), \qquad \text{Agreement} = \frac{1}{N}\sum_t \mathbf{1}[a_t^E \in \mathcal{A}_{\min}(o_t)].$$

This construction metric counts support for the demonstrated action, including ties. It differs from fixed-state action agreement with Full, which compares the evaluators' selected primitive actions.

Repair. A judgment agent examines missing expert-action support and implausible candidate actions using the local context, subgoal, demonstration, and candidate set. A predicate-gap agent decides whether an actionable case requires predicate refinement or can be handled by the existing library. A tree-repair agent proposes local patches, such as a guarded AND branch. Each patch is re-evaluated by the verifier before retention.

L.1 Construction Outcomes

Figure 16 illustrates verifier-guided refinement and structural growth in one construction. The policy evaluation in Table 17 repeats construction from scratch, then trains and evaluates one controller per generated scaffold, using the same architecture, learning procedure, and inference protocol. Its variation therefore includes both construction and training. The reference repeats training on the fixed main-experiment scaffold. The comparable policy performance supports using independently induced recursive factorizations.

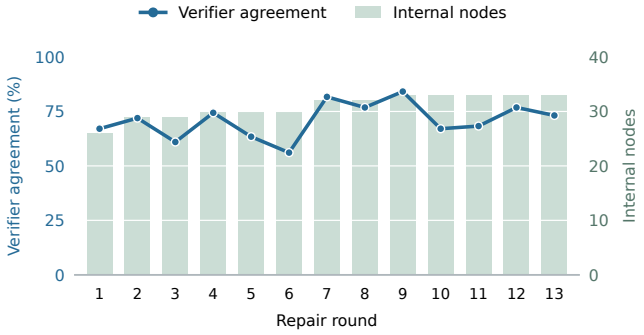


Figure 16: Verifier-guided refinement dynamics for a representative automatically generated recursive scaffold on BabyAI `UnblockPickup`. The line reports verifier agreement across successive repair rounds, defined by whether the demonstrated action belongs to the current minimum-energy candidate action set. The bars report scaffold complexity measured by the number of internal nodes. The trajectory illustrates iterative structural refinement under verifier feedback with moderate growth in scaffold complexity.

Table 17: Construction robustness on BabyAI `UnblockPickup`. Automatically induced results summarize independent constructions from scratch, each followed by one controller training and evaluation. The reference column reports six training runs with the fixed main-experiment scaffold. Values are mean±SD over the respective repetitions.

Metric	Automatically induced	Reference scaffold
Policy success	$82.0 \pm 4.4\%$	$85.67 \pm 6.35\%$
Tree nodes	31.3 ± 2.1	34

M Scaffold-Construction Prompts

We report instruction-prompt excerpts with formatting edits and terminology normalized to the verifier-agreement definition above. The task instructions and repair conditions are retained. Dynamic JSON payloads such as demonstrations, current tree JSON, predicate-library JSON, verifier counterexamples, and label jobs are not reproduced in full; the omitted payload fields are described below each 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 verifier agreement: the expert action is in the candidate set.
- 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 exact matching of one demonstrated action.
- 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.

N LLM Planner Prompts and Schemas

N.1 Stage-distillation prompt

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>
```

N.2 EXAMPLE 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."
    }
  ]
}
```


N.3 ONLINE PLANNING PROMPT

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>
```

O Extensions and Future Directions

Dual-resolution recursive energy provides a basis for learned scaffolds, action-conditioned dynamics, larger action spaces, and open-ended embodied control. The wake–sleep separation also allows reinforcement-learning objectives for action preferences while retaining expanded-energy distillation and contraction-error fitting. Extending learned-radius calibration across these settings and larger distribution shifts is a promising direction for adaptive deliberation.