Baba in Wonderland: Online Self-Supervised Dynamics Discovery for Executable World Models

SeungWon Seo*
Korea University
Seoul, Republic of Korea
ssw03270@korea.ac.kr

DongHeun Han*
Kyung Hee University
Yongin, Republic of Korea
hand32@khu.ac.kr

SeongRae Noh
Korea University
Seoul, Republic of Korea
rhosunr99@korea.ac.kr

HyeongYeop Kang†
Korea University
Seoul, Republic of Korea
siamiz_hkang@korea.ac.kr

Abstract

Executable world models can be read, edited, executed, and reused for planning, but only if the program captures the environment's transition law rather than semantic shortcuts in its surface vocabulary. We study online executable world-model learning under prior misalignment, where an agent must induce state-dependent dynamics from interaction evidence alone, without rule descriptions, reward signals, or trustworthy lexical priors. We introduce **Alice**, a closed-loop system that treats failed candidate updates as structural signal: when a candidate explains a new transition but loses previously explained ones, the preservation conflict reveals dynamics that the current program had conflated. Alice refines these conflicts into hypothesis classes that both provide a few broken examples, one per affected class, for update and guide frontier exploration toward transitions that are novel and underrepresented with respect to the current program. We evaluate Alice on *Baba in Wonderland*, a prior-misaligned variant of *Baba Is You* that preserves simulator dynamics while replacing semantically meaningful rule-property labels with unrelated words. Experiments show that Alice substantially improves executable world-model learning under prior misalignment, and five-run ablations support overall improvement trends as class structure and exploration signals are added. Project page: https://ssw03270.github.io/BABAinWonderland_NeurIPS-26_Page/.

1 Introduction

Executable world models promise dynamics models that can be read, edited, executed, and reused for planning without post-hoc interpretation [5, 19, 26, 31]. This promise, however, depends on whether the program captures the environment's transition law. When a language model synthesizes code in an environment whose surface vocabulary aligns with pretrained semantic priors, much of this inductive burden is supplied for free. When it does not, the same vocabulary becomes an active source of error [32, 40, 42], and the editable program, however clean its source, encodes the wrong dynamics.

Recent progress on executable and programmatic world models has largely operated on the favorable side of this divide, relying on curated transition traces, natural-language rule descriptions, or vocabularies whose surface meanings match the dynamics they label [5, 19, 26, 31]. A separate line

*These authors contributed equally to this work.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

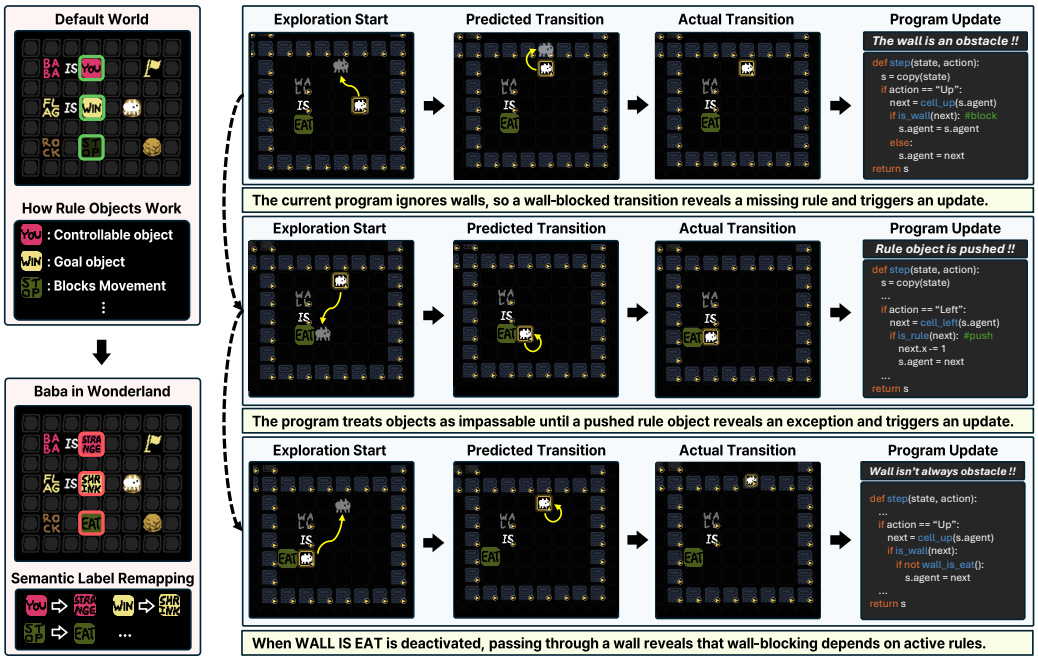

Figure 1: **Alice** in *Baba in Wonderland*. Left: simulator dynamics are preserved while rule-property labels are semantically remapped, removing lexical shortcuts. Right: exploration exposes prediction errors that drive successive executable world-model revisions.

confronts regimes without such scaffolds, but typically reasons inside the Large Language Model (LLM) rather than synthesizing a persistent program [32, 42]. Their intersection – online executable world-model construction without trustworthy surface vocabulary – has received little direct study, yet this is where executable models are most useful: novel domains, opaque simulators, and unfamiliar terminology. The agent cannot be handed the rules; it must induce them from interaction, and an opaque neural model gives up much of the reason to seek executable form.

We instantiate this regime with **Baba in Wonderland**, a controlled prior-misaligned variant of *Baba Is You*[3] The original game already exhibits state-dependent dynamics: rule sentences live inside the state, so the same action can produce different outcomes as rule blocks are rearranged. Yet labels such as `STOP`, `MOVE`, and `WIN` still provide semantic shortcuts. Baba in Wonderland preserves the simulator dynamics, action space, and rule-parsing mechanics, but replaces these labels with semantically unrelated words and provides no dictionary. A remapped token remains a stable identifier for the same role, but its surface form no longer hints at how that role behaves; successful modeling must be grounded in transition evidence rather than lexical priors.

Building an executable world model online in this regime exposes two coupled bottlenecks. The first is an *update bottleneck*: each unexplained transition demands a code revision that explains the new transition without losing previously explained ones. Providing the full history in the update prompt is impractical because the dataset grows while the LLM context window does not [22, 23, 37]. Updating from a small subset is context-efficient [38] but risks brittle revisions that fit the target while silently regressing on rare or structurally distinct cases [43]. The question is therefore which subset can expose distinctions that the program itself has not yet made.

The second is an *exploration bottleneck*. Improving the model requires transitions that expose gaps in its current hypotheses, but no external reward, rule description, or pretrained prior identifies which transitions those are [15, 32, 42]. Without such a signal, exploration either spends budget on transitions the current program already explains or traverses the state graph without knowing which transitions matter. What is needed is a notion of "novelty with respect to the program being built", derived from interaction rather than rewards or priors.

[3] Steam page for Baba Is You.

Our central observation is that both bottlenecks share a hidden source of signal: *the program's own update failures*. When a candidate revision explains a new transition but loses previously explained ones, the lost transitions are evidence that the earlier program had conflated distinct dynamics under a single branch of code. The preservation conflict reveals a latent partition of accumulated evidence into hypothesis classes that respond differently to the proposed change. Once explicit, this partition also defines what is underrepresented relative to the current program, turning a rejection event into the substrate for both compact update evidence and program-relative exploration.

We realize this idea in **Alice**, a closed-loop system that maintains a single persistent executable program while alternating LLM-based updates with update-guided exploration. Rejected candidate updates refine accumulated transitions into hypothesis classes, which serve two roles: they supply a few broken examples, one per affected class, for update, and they shape an embedding space in which a frontier explorer scores candidates by particle-entropy novelty and class-rarity coverage. Across Baba in Wonderland and the original Default World, Alice learns substantially more accurate executable world models than the closest prior interaction-driven baseline, with the gap widening sharply under prior misalignment; five-run ablations support overall improvement trends as class structure and exploration signals are added. Alice is a general method for online executable world-model learning, not a method specific to *Baba Is You*. We use *Baba in Wonderland* as a controlled probe to assess dynamics discovery from transition evidence while reducing reliance on pretrained knowledge of label meanings. We also evaluate Alice in the more realistic household setting of ALFWorld [30], extending the evaluation beyond *Baba Is You* (Table 3). Figure 1 illustrates the setting and update loop.

We summarize our contributions as follows:

- **A regime and a benchmark.** We introduce *Baba in Wonderland*, a controlled variant of *Baba Is You* in which simulator dynamics are preserved but surface labels are remapped to semantically unrelated words. As a controlled probe, the benchmark isolates online executable world-model learning in a regime where lexical priors offer no shortcut.
- **Failed updates as structural signal.** We show that rejected candidate program updates expose a latent partition of accumulated transitions into hypothesis classes, refined monotonically as more updates are attempted. This turns an event normally discarded as a rejection into a reusable artifact for organizing experience.
- **A closed loop that uses this signal twice.** The same hypothesis classes supply a few broken examples, one per affected class, for the next program update and shape an embedding in which a frontier explorer scores candidates by novelty and class rarity. One signal therefore drives both program updates and data collection.
- **Empirical evidence under prior misalignment.** On Baba in Wonderland, Alice substantially outperforms the closest interaction-driven executable world-model baseline; five-run ablations assess overall trends as class structure and exploration signals are added.

2 Related Work

Learning executable world models. World-model learning is often studied through neural latent dynamics models [8–11], and recent work also uses LLMs as implicit simulators or reasoning backends for planning [12, 14, 28, 29, 36]. These models can be powerful, but their dynamics are not preserved as a persistent executable artifact that can be inspected, locally updated, and reused as source code. This distinction matters in environments where pretrained semantic priors conflict with the transition law [32, 40, 42]: language-space reasoning can follow surface meanings even when those meanings are misleading or when dynamics vary with state.

The closest line of work asks LLMs to externalize environment knowledge as executable code or structured programmatic world models, including reusable procedures for embodied control [20, 35], interaction-driven code world models [31], search-guided synthesis [5], natural-language-rule-to-code game models [19], and compositional code-based modeling [26]. Our work shares the goal of learning an executable world model, but studies a stricter regime: the agent must maintain a persistent program online from interaction evidence alone, while surface vocabulary is prior-misaligned and the effective transition law changes with the state. Baba Is AI [4] and Baba is Y'all 2.0 [3] study rule manipulation and collaborative level design, respectively. Our benchmark's novelty lies in the intervention rather

than the game: it remaps only the surface labels while keeping the transition law and interaction protocol fixed, providing a controlled probe of dependence on lexical priors. Other studies probe post hoc whether foundation models have learned world structure [33] or develop world models for agent interactions [39, 41]. Unlike these, Alice constructs a persistent executable model online and uses failed program updates to reveal distinctions between transition dynamics, reusing this evidence for both program refinement and exploration.

Self-supervised exploration for dynamics discovery. Online world-model learning also requires exploration: the agent must collect transitions that expose gaps in the current hypothesis rather than merely expand a dataset. Task reward can direct data collection in model-based control [10], and solution-finding objectives can guide executable world-model agents [31], but these objectives may be unavailable in reward-free settings [15] or misaligned with dynamics discovery. Intrinsic-motivation and reward-free methods instead use information gain or uncertainty [13, 17], novelty [2, 24], disagreement [25], memory or directed exploration [1, 6], and entropy or mutual information in behavior and transition spaces [7, 18, 21, 34].

For executable world-model learning, however, generic diversity is not enough: exploration should be relative to the program being updated. Some prior work makes exploration model-relative through learning progress [17] or anticipated model novelty [27], but these signals are designed for neural predictors or downstream control adaptation. Alice instead derives the exploration space from the executable update process itself. Rejected candidate updates induce hypothesis classes over transition evidence, and the Explorer uses these classes to measure novelty and underrepresentation with respect to the current program's explanatory frontier.

3 Problem Setting

3.1 Formal Setup

We formulate online executable world-model learning in a deterministic Markov decision process (MDP). Let $\mathcal{S}$ and $\mathcal{A}$ denote discrete state and action spaces, and let $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denote the environment transition function. At each global interaction step t, the agent observes a transition $\tau_t = (s_t, a_t, s_{t+1})$ with $s_{t+1} = T(s_t, a_t)$, and accumulates the dataset $D_t = (\tau_1, \tau_2, \ldots, \tau_t)$.

Rather than learning a neural predictor, we represent the world model as a source-code program P. We interpret P as an executable hypothesis about T, whose validity is tested only through observed transition evidence: given (s, a), the program either returns a predicted next state $\hat{s}'$ or fails to execute. We say that P explains a transition $\tau = (s, a, s')$ if it executes successfully on (s, a) and outputs $\hat{s}' = s'$, and we write $E(P; D) = \{\tau \in D \mid P \text{ explains } \tau\}$ for the explained subset of D.

The agent operates under a finite global interaction budget H, and the executable hypothesis is continually revised from the accumulated evidence in D_t. Let P_i denote the current accepted program, with i indexing accepted updates. When a newly observed transition τ_{t+1} is not explained by P_i, the agent proposes a candidate update Q, which is accepted as P_{i+1} only if it explains τ_{t+1} while preserving every previously explained transition: $E(P_i; D_t) \cup \{\tau_{t+1}\} \subseteq E(P_{i+1}; D_{t+1})$. The system-level objective is to use the interaction budget H to expand $E(P_i; D_t)$, and exploration contributes by selecting interactions likely to expose explanatory failures of P_i.

This is not an offline modeling problem over a fixed transition dataset. In our setting, pre-collected traces and rule descriptions are unavailable, and task rewards do not specify which transitions improve the hypothesis. Exploration therefore means collecting online evidence for refining the current executable hypothesis about T, not maximizing external reward.

3.2 Baba in Wonderland

We instantiate this setting in *Baba Is You*, a puzzle game in which rule sentences live inside the grid as movable text blocks. The environment is challenging on two fronts. Its state space is large and combinatorial, so unguided exploration cannot enumerate it within any realistic budget. More fundamentally, because rules are part of the state, the same action can produce different outcomes depending on which sentences are currently active. A correct hypothesis about T must therefore capture how the currently active rule configuration determines transition outcomes — not a fixed background dynamics, but a transition law that the agent's own actions can rewrite.

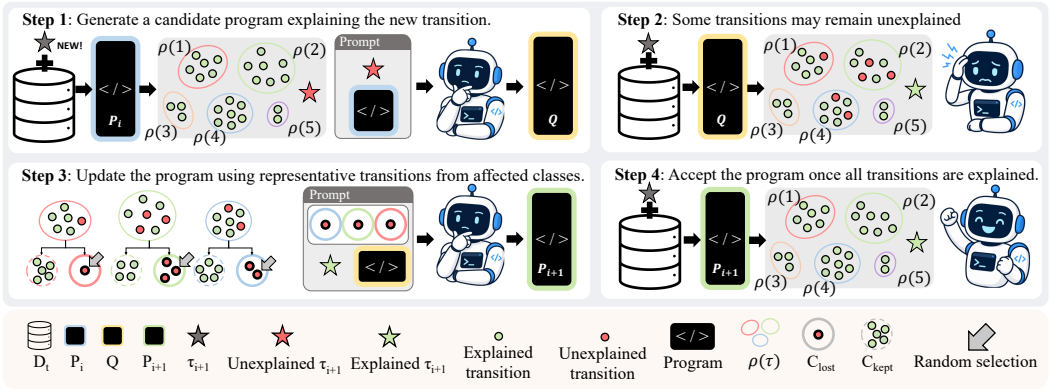

Figure 2: Hypothesis-class refinement. Rejected target-explaining updates reveal lost transitions, split affected classes, and provide broken examples for later retries.

The original game, however, has one feature that obscures the difficulty we want to study. Its rule-property labels — `STOP`, `MOVE`, `WIN` — still carry their everyday meanings, so an LLM can often guess correct dynamics from surface vocabulary alone. This residual semantic shortcut conflates two distinct sources of success: induction from transition evidence, and lookup from pretrained priors. To separate them, we introduce *Baba in Wonderland*, a minimal intervention that preserves the simulator dynamics, action space, and rule-parsing mechanics, and changes only the surface labels of `rule_property` tokens, remapping them to semantically unrelated words; see Appendix A.1 for environment details and the full remapping.

The agent receives no dictionary and no natural-language explanation of the remapping. The role of each remapped token must therefore be inferred from transition evidence and from how the token participates in active rule sentences, rather than from pretrained lexical priors alone.

4 Method

Section 3 identified two coupled bottlenecks for online executable world-model learning: an *update bottleneck*, in which each candidate revision must preserve every previously explained transition under a context budget that cannot hold all of D_t, and an *exploration bottleneck*, in which no external signal indicates which transitions would expose gaps in the current hypothesis. We propose **Alice**, a closed-loop system that addresses both by treating the same artifact — failed candidate updates — as a structural signal: the conflicts that arise when a candidate gains τ_{t+1} but loses some previously explained transitions reveal which transitions belonged to distinct hypotheses all along.

The system has two components that share this signal. The **Programmer** maintains the persistent program P_i and uses preservation conflicts to organize D_t into hypothesis classes that supply compact broken examples for the next update. The **Explorer** uses the same classes to score frontier candidates by both novelty and class underrepresentation, steering data collection toward the program's explanatory frontier. The two components operate in a closed loop, alternating program updates with update-guided exploration over the agent's interaction budget H. Algorithm 1 summarizes this loop.

4.1 Refining Hypothesis Classes from Failed Updates

When τ_{t+1} is unexplained by P_i, the Programmer asks an LLM to propose a candidate update Q that explains τ_{t+1} while preserving every transition in $E(P_i; D_t)$. Verifying preservation is straightforward: re-run Q on every transition in D_t and check that the explained set still contains $E(P_i; D_t)$. *Generating* a candidate that will pass this check is harder, because the prompt must convey what those transitions actually look like, and neither P_i nor τ_{t+1} alone is enough. P_i is an executable hypothesis induced from D_t, not a lossless record of it: its code specifies how to compute next states without recording which transitions support each branch. The evidence must therefore come from D_t itself — but D_t grows online while the prompt budget is fixed, so the Programmer cannot include it in full.

The Programmer therefore selects a compact reference set $R_t \subseteq D_t$ to include in the prompt. The full inductive program-update prompt template is provided in Appendix A.2. The question is which transitions should be in R_t.

The answer is that the LLM itself can produce this evidence whenever a candidate is rejected. Suppose Q explains τ_{t+1} but fails to preserve some previously explained transitions. We call the set of lost transitions the *broken examples* of Q,

$$L_{\mathrm{pc}}(Q; P_i, D_t) = E(P_i; D_t) \setminus E(Q; D_t).$$

These are transitions that P_i explained, grouping them with whatever rule its code applies in the relevant branch, yet Q now explains them differently from τ_{t+1}. The merger inside P_i was therefore hiding a distinction, and L_{pc} surfaces exactly the transitions where that distinction matters. Each rejected candidate that explains its target thus contributes one cut through $E(P_i; D_t)$, separating transitions that respond differently to the proposed update. The Programmer accumulates these cuts as a mutable collection of *hypothesis classes* $\mathcal{C}$ over $E(P_i; D_t)$. Transitions remain in the same class until separated by a rejected candidate (Figure 2).

We initialize $\mathcal{C}$ from the accepted update sequence itself. Transitions that entered the explained set under the same bundle of hypotheses tend to provide redundant preservation evidence, so we group them by the earliest accepted version from which they have remained explained:

$$\rho(\tau) = \min\{j \le i \mid \tau \in E(P_k; D_t) \text{ for all } k = j, \ldots, i\}.$$

For example, if τ is explained by P_1, lost by P_2, and explained by every version from P_3 onward, then $\rho(\tau) = 3$. Transitions sharing $\rho(\tau)$ form the *root classes* of $\mathcal{C}$. Here, earlier program versions are evaluated retrospectively on the current dataset.

Root classes alone are coarse, however, because an early update can explain its target with a rule broader than the truth, sweeping in transitions that should later prove to obey something different. Rejected candidates expose such overgeneralizations at the class level. When $L_{\mathrm{pc}}(Q; P_i, D_t)$ covers only part of a class C, the candidate Q separates C into a preserved subset and a lost subset,

$$C_{\mathrm{lost}} = C \cap L_{\mathrm{pc}}(Q; P_i, D_t), \qquad C_{\mathrm{kept}} = C \setminus C_{\mathrm{lost}},$$

giving evidence that C had merged transitions responding differently to the new hypothesis. Whenever both subsets are nonempty, the Programmer replaces C with the two refined classes and stores Q as the split-inducing test that will route future transitions to the appropriate side. A refined class can be split again by later rejected candidates, making $\mathcal{C}$ monotonically finer over retries.

To build the small reference set R_t, the Programmer groups the broken examples by their refined classes, chooses up to $n = 3$ affected classes, and samples $m = 1$ broken example from each. These few transitions, each witnessing a distinct preservation constraint, replace what would otherwise be random samples from D_t. The acceptance check itself remains over the full $E(P_i; D_t)$, so R_t shapes only the update context, not the correctness criterion. After Q is accepted, τ_{t+1} and later explained transitions inherit the existing refinements: they are routed first by ρ into their root class, then by the stored split candidates into the appropriate refined class, so newly explained transitions immediately contribute evidence to the same structure that produced them. The same $\mathcal{C}$ also guides how the Explorer collects future transitions in Section 4.2.

4.2 Update-Guided Exploration

The Programmer turns transitions into program updates; the Explorer must turn the interaction budget H into the informative transitions. A transition is informative for the next program update if two conditions hold: it should expose a failure of the current program P_i so that an update is triggered, and ideally it should also induce a new hypothesis class so that $\mathcal{C}$ refines further. Neither property is observable before the candidate is executed. The Explorer therefore maintains a transition graph over reached states and scores each frontier candidate $x = (s, a)$, where s is reached but unexpanded, using two pre-execution proxies: *embedding novelty*, which favors candidates lying far from D_t in a learned state-action space, and *expected class coverage*, which favors candidates likely to fall into hypothesis classes with little existing evidence. Together, these proxies bias graph-frontier exploration toward the program's explanatory frontier. Figure 3 illustrates the procedure.

For embedding novelty to track the program's explanatory frontier rather than mere surface variation, candidates that fall in the same hypothesis class should map to nearby embeddings, and candidates

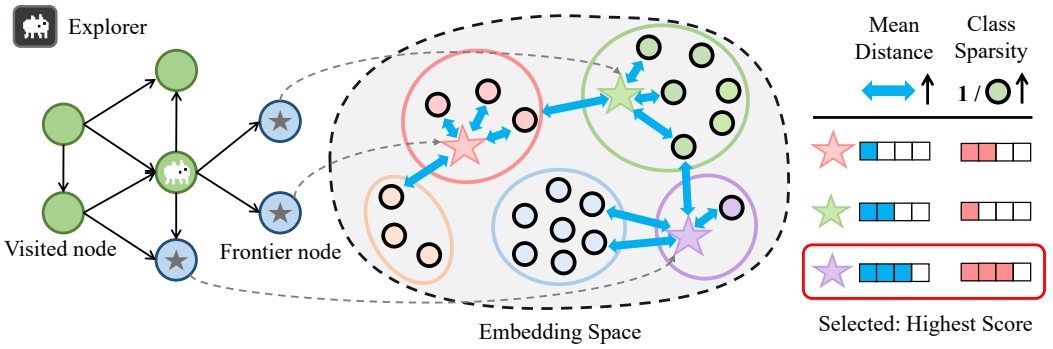

Figure 3: Update-guided frontier scoring. The Explorer selects frontier candidates by combining embedding novelty with expected coverage of underrepresented hypothesis classes.

from different classes should map to distant ones. The hypothesis classes $\mathcal{C}$ provide exactly the supervision needed for this geometry: the Explorer trains $h_\theta(x)$ with the supervised contrastive loss of Khosla et al. [16], pulling together embeddings within the same class and pushing apart embeddings across classes. Supervision is limited to active classes $\mathcal{C}_{\text{act}} \subseteq \mathcal{C}$ with enough transitions for stable positives; low-count classes remain in $\mathcal{C}$ but are excluded.

Given this embedding, novelty is measured by a k-nearest-neighbor particle-entropy score [21],

$$r_h(x) = \log\left(1 + \frac{1}{k} \sum_{\tilde{h} \in \mathcal{N}_k(h_\theta(x); \mathcal{B}_h)} \|h_\theta(x) - \tilde{h}\|_2\right),$$

where $\mathcal{B}_h$ is the embedding set induced by D_t and $\mathcal{N}_k$ returns the $k = 32$ nearest neighbors. Candidates far from previously observed embeddings score high.

The second proxy, *expected class coverage*, biases exploration toward classes with little existing evidence. Using the same active class set, each $C \in \mathcal{C}_{\text{act}}$ contributes a class-count self-information $u(C) = -\log(|C|/N)$, where $N = \sum_{C \in \mathcal{C}_{\text{act}}} |C|$ is the total count across active classes; rare classes have larger $u(C)$. Since the class label of an unexecuted candidate is itself unobservable, the Explorer estimates it from class prototypes — up to 8 representative embeddings per class. A softmax over cosine similarities to prototype embeddings yields a pre-execution class distribution $q_\theta(C \mid h_\theta(x))$, and the expected class-coverage score is

$$r_{\mathcal{C}}(x) = \sum_{C \in \mathcal{C}_{\text{act}}} q_\theta(C \mid h_\theta(x))\, u(C).$$

The two proxies combine into the frontier score

$$S_{\text{frontier}}(x) = \lambda_h r_h(x) + \lambda_{\mathcal{C}} r_{\mathcal{C}}(x).$$

with $\lambda_h = 1$ and $\lambda_{\mathcal{C}} = 0.35$ in the main online experiments; Appendix A.3 reports the sensitivity analysis. Each collection batch greedily selects the highest-scoring candidates from the current frontier under the interaction budget.

Executed transitions feed back into the loop. Explained transitions sharpen the class-structured evidence used to train h_θ, while unexplained transitions become the Programmer's next explanatory failures. The single signal of failed updates therefore drives both halves of Alice: it shapes which broken examples the LLM sees in R_t, and it shapes which transitions the agent seeks through S_{frontier}. Appendix A.4 relates this score to an entropy–mutual-information objective.

5 Experiments

We evaluate Alice along two axes that jointly support the claims in Section 1: (i) end-to-end accuracy of the learned executable world model under prior misalignment, both for online interaction on

Table 1: Online interaction results on regular levels. All Acc. and Balanced Acc. are exact one-step transition accuracies; Calls reports learning-time LLM calls. Five-run mean $\pm$ standard deviation.

	Default World			Baba in Wonderland		
Method	All Acc. ↑	Balanced Acc. ↑	Calls	All Acc. ↑	Balanced Acc. ↑	Calls
WorldCoder	0.871 ± 0.054	0.727 ± 0.160	61.8 ± 21.5	0.542 ± 0.113	0.260 ± 0.126	57.6 ± 5.5
Alice	0.988 ± 0.008	0.990 ± 0.004	19.8 ± 6.6	0.965 ± 0.054	0.951 ± 0.068	94.0 ± 10.4

regular levels and for held-out generalization to extra levels; and (ii) ablations assessing trends as components are added. Robustness to LLM backbones, additional diagnostic experiments, and qualitative examples are reported in Appendix A.5 and Appendix A.6.

5.1 Experimental Setup

Settings. We evaluate executable world-model learning on levels drawn from *Baba Is You*, comprising 32 regular levels and 8 extra levels. Each level is instantiated under two label settings: *Default World*, which uses the original `rule_property` labels, and *Baba in Wonderland*, which preserves the simulator dynamics but replaces the surface labels of `rule_property` tokens with semantically unrelated labels (Table 7). The agent is given no dictionary and no description of the remapping. Online experiments test learning from self-collected evidence: they collect interaction data on regular levels and evaluate on human-play solution transitions from the same levels. Offline experiments test generalization from a fixed dataset to unseen levels: they train on regular-level solution transitions and evaluate on held-out extra-level transitions, so their accuracies are not directly comparable with the online ones. Extra-level evaluation of the online programs is reported in Appendix A.3. Additional protocol details appear in Appendix A.7.

To evaluate a more realistic domain, we use ALFWorld's household tasks [30] with a fully observable symbolic state/action interface following WorldCoder [31]. We train on all six task types without a curriculum and evaluate exact next-state prediction on unseen episodes. Each method receives up to 100,000 interaction steps before a program update, with the same 15-call update budget and 100-call run cap used in Baba. Appendix A.8 provides further protocol details.

Metrics. *All Acc.* is the fraction of evaluated transitions for which the predicted next state exactly matches the environment next state. *Balanced Acc.* applies the same exact-match criterion on a class-reduced evaluation set in which transitions are grouped by action and canonical state-difference signature, with one representative sampled per group; details appear in Appendix A.9. Compile failures and runtime errors are counted as incorrect predictions. We additionally report *Calls*, the cumulative number of LLM calls used during learning. For online runs each method is capped at 100 total LLM calls and 15 calls per single update iteration; offline runs are similarly capped at 100 total LLM calls.

Baselines. For online experiments, we compare against **WorldCoder** [31], the closest prior interaction-driven executable world-model learner. Like Alice, WorldCoder maintains executable code as its world model, but it does not organize failed program updates into hypothesis classes for later preservation or exploration. For offline fixed-data experiments, we additionally compare against **GIF-MCTS** [5] and **CWM** [19], which together cover search-based and population-based code world-model synthesis. All baselines use the same LLM backbone (GPT-5.4) and the same LLM-call caps as Alice; in the offline comparison Alice trains only on the fixed dataset and does not use its Explorer. The main comparisons and ablations report mean $\pm$ standard deviation over five independent runs.

5.2 Executable World-Model Learning

Online interaction. Table 1 reports online performance on regular levels in both label settings. In *Default World*, WorldCoder remains relatively strong but trails Alice in both All Acc. and Balanced Acc., reflecting that semantic priors help with many common transitions but can still miss rare or compositional dynamics. The gap widens sharply in *Baba in Wonderland*, where lexical priors are

Table 2: Offline fixed-data results. Methods train on regular-level solution transitions and evaluate on held-out extra-level transitions. Five-run mean ± standard deviation.

	Default World			Baba in Wonderland		
Method	All Acc. ↑	Balanced Acc. ↑	Calls	All Acc. ↑	Balanced Acc. ↑	Calls
GIF-MCTS	0.853 ± 0.046	0.754 ± 0.151	100.0 ± 0.0	0.796 ± 0.065	0.716 ± 0.061	100.0 ± 0.0
CWM	0.964 ± 0.047	0.934 ± 0.058	100.0 ± 0.0	0.845 ± 0.051	0.702 ± 0.149	100.0 ± 0.0
Alice	1.000 ± 0.000	1.000 ± 0.000	21.4 ± 1.5	0.948 ± 0.038	0.920 ± 0.047	91.8 ± 8.8

Table 3: Exact next-state prediction on unseen ALFWorld episodes. The third row adds a curriculum. Five-run mean ± standard deviation.

Method	Accuracy ↑	Calls
Alice	0.962 ± 0.024	34.4 ± 7.9
WorldCoder	0.702 ± 0.077	100.0 ± 0.0
WorldCoder + curriculum	0.855 ± 0.027	100.0 ± 0.0

unreliable: WorldCoder's All Acc. drops by 0.329 and its Balanced Acc. falls to 0.260, while Alice retains 0.965 All Acc. and 0.951 Balanced Acc.

The WorldCoder call counts deserve attention. In *Baba in Wonderland*, WorldCoder uses 57.6 calls on average, partly because its update loop terminates after exhausting the per-update retry budget. Allowing it to continue to the same 100-call cap raises its All Acc. to 0.819 ± 0.052 and Balanced Acc. to 0.658 ± 0.131, still below Alice (Appendix A.3). Alice addresses this failure mode by supplying the LLM with a few broken examples that target the constraints a naive update tends to violate.

Fixed-data comparison. Table 2 compares Alice with GIF-MCTS and CWM when all methods learn from the same fixed regular-level transitions and are evaluated on held-out extra-level transitions. In *Default World*, semantically aligned surface labels make executable-rule synthesis easier for all methods, and Alice reaches perfect held-out accuracy with 21.4 LLM calls on average. CWM is the strongest baseline in this setting, yet its remaining Balanced Acc. gap indicates that high overall accuracy can still miss less frequent transition types. The comparison becomes more diagnostic in *Baba in Wonderland*: because no method can collect additional transitions from the unseen extra levels, held-out accuracy depends on how effectively each method converts the fixed train evidence into executable dynamics without relying on lexical shortcuts. Here Alice improves over both GIF-MCTS and CWM at a weighted token cost between theirs (Appendix A.10), with the largest margin in Balanced Acc., showing that the persistent update loop better preserves rare or easily conflated dynamics under prior misalignment.

World-model learning in ALFWorld. Table 3 evaluates Alice on realistic household tasks. Alice achieves 0.962 ± 0.024 accuracy with 34.4 ± 7.9 calls, compared with WorldCoder's 0.702 ± 0.077 at the 100-call cap. Adding a six-task curriculum improves WorldCoder to 0.855 ± 0.027, indicating that task ordering explains part of the gap. Alice reaches higher accuracy without a curriculum, demonstrating applicability beyond rule-manipulation puzzles.

5.3 Ablation Study Results

Hypothesis classes. We ask how much structure is needed for the Programmer to turn broken examples into accepted program updates. **Single-Class** collapses all explained transitions into one class, while **Root-Class** keeps only the root classes induced by $\rho(\tau)$, before any refinement from rejected candidates. Balanced accuracy rises from 0.829 with Single-Class to 0.906 with Root-Class and reaches 0.951 with Alice (Table 4). Statistical tests support this overall increasing trend (Appendix A.3). This pattern supports the role of failed updates in separating preservation constraints that remain conflated within coarse root classes. The resulting class-stratified R_t exposes these distinctions to the Programmer with a few broken examples.

Exploration objective. We ask whether the Explorer must actively shape which transition evidence is collected before each program update. **BFS** selects frontier candidates in breadth-first order,

Table 4: Online ablations on regular-level Wonderland solution transitions. Five-run mean $\pm$ standard deviation.

Setting	All Acc. ↑	Balanced Acc. ↑	Calls
Single-Class	0.890 ± 0.053	0.829 ± 0.073	84.8 ± 20.9
Root-Class	0.940 ± 0.052	0.906 ± 0.071	80.0 ± 12.8
BFS	0.901 ± 0.071	0.854 ± 0.105	86.8 ± 18.6
w/o r_h	0.953 ± 0.044	0.924 ± 0.053	83.0 ± 13.5
w/o r_C	0.929 ± 0.050	0.890 ± 0.062	87.0 ± 20.5
Alice	0.965 ± 0.054	0.951 ± 0.068	94.0 ± 10.4

removing program-relative frontier scoring; **w/o** r_h and **w/o** r_C remove the embedding-novelty and expected-class-coverage terms from S_{frontier}, respectively. Both score ablations have higher mean accuracy than BFS but lower than Alice. This pattern is consistent with complementary roles for the two score terms: r_h favors novelty in the learned state-action space, while r_C directs exploration toward underrepresented hypothesis classes. Their combination helps collect evidence that is both novel and relevant to weakly supported parts of the current program. See Appendix A.3 for significance tests. Learning curves and embedding visualizations provide additional analyses in Appendix A.5.

6 Conclusion

We studied online executable world-model learning under prior misalignment, where state-dependent dynamics must be induced from interaction evidence rather than rule descriptions, rewards, or trustworthy lexical priors. Alice maintains a persistent executable program and turns failed candidate updates into hypothesis classes that support both class-stratified program update and program-relative frontier exploration. Through *Baba in Wonderland*, we show that this closed loop can ground executable world-model construction in transition evidence even when surface labels are misleading. Experiments on ALFWorld demonstrate Alice's applicability to realistic household tasks beyond rule-manipulation puzzles.

However, our experiments are limited to deterministic, discrete, fully observable symbolic environments where transition explainability can be checked exactly by executing a program. Full-set checking also incurs increasing local computation as evidence accumulates, as detailed in Appendix A.10. Extending the idea to stochastic, continuous, or partially observable domains will require broader notions of explanation, preservation, and update-induced evidence.

Acknowledgments and Disclosure of Funding

This work was supported by the ICT Creative Consilience Program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (IITP-2026-RS-2020-II201819), and by the IITP grant funded by the Korea government (MSIT) (IITP-2026-RS-2026-25615490, AI Star Fellowship Support Program). The authors declare no competing interests.

References

[1] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

[2] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[3] Megan Charity, Isha Dave, Ahmed Khalifa, and Julian Togelius. Baba is y'all 2.0: Design and investigation of a collaborative mixed-initiative system. *IEEE Transactions on Games*, 16(1): 75–89, 2022.

[4] Nathan Cloos, Meagan Jens, Michelangelo Naim, Yen-Ling Kuo, Ignacio Cases, Andrei Barbu, and Christopher J Cueva. Baba is ai: Break the rules to beat the benchmark. *arXiv preprint arXiv:2407.13729*, 2024.

[5] Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. *Advances in Neural Information Processing Systems*, 37:60429–60474, 2024.

[6] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[7] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[8] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

[9] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[10] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[11] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

[12] Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, 2023.

[13] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.

[14] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

[15] Chi Jin, Akshay Krishnamurthy, Max Simchowitz, and Tiancheng Yu. Reward-free exploration for reinforcement learning. In *International Conference on Machine Learning*, pages 4870–4879. PMLR, 2020.

[16] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/d89a66c7c80a29b1bdbab0f2a1a94af8-Abstract.html.

[17] Kuno Kim, Megumi Sano, Julian De Freitas, Nick Haber, and Daniel Yamins. Active world model learning with progress curiosity. In *International conference on machine learning*, pages 5306–5315. PMLR, 2020.

[18] Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. Unsupervised reinforcement learning with contrastive intrinsic control. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 34478–34491. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/debf482a7dbdc401f9052dbe15702837-Paper-Conference.pdf.

[19] Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou, Carter Wendelken, Zun Li, Antoine Dedieu, Jordi Grau-Moya, Marc Lanctot, Atil Iscen, et al. Code world models for general game playing. *arXiv preprint arXiv:2510.04542*, 2025.

[20] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[21] Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34:18459–18473, 2021.

[22] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

[23] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

[24] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[25] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

[26] Wasu Top Piriyakulkij, Yichao Liang, Hao Tang, Adrian Weller, Marta Kryven, and Kevin Ellis. Poe-world: Compositional world modeling with products of programmatic experts. *arXiv preprint arXiv:2505.10819*, 2025.

[27] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International conference on machine learning*, pages 8583–8592. PMLR, 2020.

[28] SeungWon Seo, SeongRae Noh, Junhyeok Lee, SooBin Lim, Won Hee Lee, and HyeongYeop Kang. Reveca: Adaptive planning and trajectory-based validation in cooperative language agents using information relevance and relative proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23295–23303, 2025.

[29] SeungWon Seo, SooBin Lim, SeongRae Noh, Haneul Kim, and HyeongYeop Kang. From assumptions to actions: Turning llm reasoning into uncertainty-aware planning for embodied agents. *arXiv preprint arXiv:2602.04326*, 2026.

[30] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[31] Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Advances in Neural Information Processing Systems*, 37:70148–70212, 2024.

[32] Xiaojuan Tang, Jiaqi Li, Yitao Liang, Song-chun Zhu, Muhan Zhang, and Zilong Zheng. Mars: Situated inductive reasoning in an open-world environment. *Advances in Neural Information Processing Systems*, 37:17830–17869, 2024.

[33] Keyon Vafa, Peter G Chang, Ashesh Rambachan, and Sendhil Mullainathan. What has a foundation model found? using inductive bias to probe for world models. *arXiv preprint arXiv:2507.06952*, 2025.

[34] Shanchuan Wan, Yujin Tang, Yingtao Tian, and Tomoyuki Kaneko. Deir: efficient and robust exploration through discriminative-model-based episodic intrinsic rewards. *arXiv preprint arXiv:2304.10770*, 2023.

[35] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[36] Ruoyao Wang, Graham Todd, Ziang Xiao, Xingdi Yuan, Marc-Alexandre Côté, Peter Clark, and Peter Jansen. Can language models serve as text-based world simulators? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–17, 2024.

[37] Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, et al. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025.

[38] Chunqiu Steven Xia and Lingming Zhang. Keep the conversation going: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. *arXiv preprint arXiv:2304.00385*, 2023.

[39] Zikai Xiao, Jianhong Tu, Chuhang Zou, Yuxin Zuo, Zhi Li, Peng Wang, Bowen Yu, Fei Huang, Junyang Lin, and Zuozhu Liu. Webworld: A large-scale world model for web agent training, 2026. *URL https://arxiv. org/abs/2602.14721*.

[40] Manjie Xu, Isabella Yin, Xinyi Tu, Chi Zhang, and Yixin Zhu. Code over words: Overcoming semantic inertia via code-grounded reasoning. *arXiv preprint arXiv:2601.18352*, 2026.

[41] Xiao Yu, Baolin Peng, Ruize Xu, Yelong Shen, Pengcheng He, Suman Nath, Nikhil Singh, Jiangfeng Gao, and Zhou Yu. Reinforcement world model learning for llm-based agents. *arXiv preprint arXiv:2602.05842*, 2026.

[42] Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. Wall-e 2.0: World alignment by neurosymbolic learning improves world model-based llm agents. *arXiv preprint arXiv:2504.15785*, 2025.

[43] Xin Zhou, Bowen Xu, Kisub Kim, DongGyun Han, Thanh Le-Cong, Junda He, Bach Le, and David Lo. Patchzero: Zero-shot automatic patch correctness assessment. *arXiv preprint arXiv:2303.00202*, 2023.

A Appendix

A.1 Baba Is You

Game rules and dynamics. Baba Is You is a rule-manipulation puzzle game in which object behavior is determined by textual rules assembled directly in the grid world. Rules become active when text blocks form valid horizontal or vertical rule sentences, and any change in that arrangement can activate, deactivate, or modify the effective dynamics of the environment. Because these rule sentences can be created, broken, moved, or rearranged by the agent's actions, the currently active rule set is itself part of the environment state rather than a fixed background specification. As a result, predicting the next state requires more than tracking the visible arrangement of objects: the same action can lead to different outcomes depending on which rules are currently active. This makes the game a particularly challenging environment for world-model learning, because its transition law is state-dependent and symbolic rather than fixed and purely geometric. In *Default World*, for example, changing whether a sentence such as `WALL IS STOP` is active can immediately alter whether walls behave as obstacles, even when the surrounding object layout remains unchanged. Baba Is You therefore provides a natural testbed for executable world-model learning in environments where accurate prediction depends on maintaining explicit hypotheses about mutable rules.

Table 5: Token vocabulary used in the *Baba Is You* benchmark environment.

Type	Vocabulary
`rule_noun`	`ALGAE`, `BABA`, `BOG`, `BOLT`, `BRICK`, `BUBBLE`, `COG`, `CRAB`, `DOOR`, `FLAG`, `FLOWER`, `GRASS`, `HEDGE`, `ICE`, `JELLY`, `KEKE`, `KEY`, `LAVA`, `LOVE`, `PILLAR`, `PIPE`, `REED`, `ROCK`, `ROBOT`, `SKULL`, `STAR`, `TEXT`, `TILE`, `WALL`, `WATER`
`rule_operator`	`AND`, `IS`
`rule_property`	`DEFEAT`, `FLOAT`, `HOT`, `MELT`, `MOVE`, `OPEN`, `PUSH`, `SHUT`, `SINK`, `STOP`, `WIN`, `YOU`
`world_object`	`ALGAE`, `BABA`, `BOG`, `BOLT`, `BRICK`, `BUBBLE`, `COG`, `CRAB`, `DOOR`, `FLAG`, `FLOWER`, `GRASS`, `HEDGE`, `ICE`, `JELLY`, `KEKE`, `KEY`, `LAVA`, `LOVE`, `PILLAR`, `PIPE`, `REED`, `ROCK`, `ROBOT`, `SKULL`, `STAR`, `TILE`, `WALL`, `WATER`

Table 6: Rule properties and their roles in the simulator used by the benchmark.

Property	Role
`YOU`	Makes an object controllable during the agent-driven movement stage.
`WIN`	Triggers success when a `YOU` object overlaps a winning object on the same float layer after interaction effects are resolved.
`STOP`	Blocks entry and movement through the object.
`PUSH`	Allows the object to be pushed by movers.
`MOVE`	Causes the object to move automatically each turn, reversing direction if blocked.
`DEFEAT`	Destroys overlapping `YOU` objects on the same float layer.
`SINK`	Removes both itself and other overlapping same-layer objects.
`HOT`	Destroys overlapping `MELT` objects on the same float layer.
`MELT`	Is removed when overlapping a `HOT` object on the same float layer.
`OPEN`	Cancels with `SHUT`, removing both on overlap.
`SHUT`	Cancels with `OPEN`, removing both on overlap.
`FLOAT`	Restricts many overlap-triggered interactions to objects on the same float layer.

Vocabulary. We use four token types throughout the benchmark: `rule_noun`, `rule_operator`, `rule_property`, and `world_object`. Table 5 summarizes the vocabulary used by each type, and Table 6 summarizes the rule properties and their roles in the simulator. Rule nouns and world objects share most of their lexical stems, but they play different semantic roles: the former appear as text blocks in rule sentences, whereas the latter are instantiated entities in the grid world. Among rule operators, `IS` links an object noun to a property or another noun, while `AND` allows conjunctions on either side of a rule, enabling forms such as `BABA AND ROCK IS YOU` and `BABA IS YOU AND PUSH`. Crucially, rule properties are not merely labels. They alter movement, collision, overlap, destruction, and goal conditions, and multiple active properties can interact within the same turn. These interactions are also order-sensitive. In the original game, their effects are resolved over an ordered update procedure rather than all at once. Broadly, movement-related properties such as

Table 7: Rule-property remapping used in *Baba in Wonderland*. Simulator semantics remain unchanged.

Original	Wonderland	Original	Wonderland	Original	Wonderland	Original	Wonderland
DEFEAT	WAKE	FLOAT	WRONG	HOT	GRIN	MELT	CURIOUS
MOVE	DRINK	OPEN	MAD	PUSH	GROW	SHUT	LATE
SINK	BEGIN	STOP	EAT	WIN	SHRINK	YOU	STRANGE

YOU, MOVE, and PUSH shape how objects are displaced, whereas overlap-triggered properties such as DEFEAT, SINK, HOT/MELT, OPEN/SHUT, and WIN are checked later in the turn. WIN is evaluated especially late, so earlier destructive effects can prevent a win that would otherwise occur. FLOAT further constrains many of these interactions by requiring the relevant objects to occupy the same float layer. As a result, learning an accurate executable world model requires not only inferring which properties are currently active, but also capturing how their effects compose under the simulator's execution order.

State and action representation. We provide states to the LLM as structured JSON rather than as free-form natural-language descriptions. Each state contains the top-level fields `grid_size`, `step`, and `objects`. `grid_size` gives the width and height of the playable interior grid, after trimming the immutable outer border used by the underlying simulator. The `step` field records episode-level status information and currently includes a Boolean `terminated` flag. Each entry in `objects` is a dictionary with keys such as `type`, `word`, and `position`, and directional objects additionally include a `direction` field. Positions are represented as zero-indexed $[x, y]$ coordinates in the trimmed playable grid. Because multiple objects may occupy the same grid cell, overlap is represented by multiple object entries sharing the same `position`. World objects are directional entities in the underlying simulator, and serialized entries expose this through a `direction` field when that orientation is available. For example, a text block may be represented as `{"type": "rule_noun", "word": "baba", "position": [0,0]}`, while a `world_object` entry may additionally include a field such as `"direction": "facing right"`.

Actions are represented by a small discrete set of named commands, `idle`, `up`, `right`, `down`, and `left`. Transitions are therefore recorded as structured triples consisting of a source-state JSON string, an action name, and a next-state JSON string.

Baba in Wonderland remapping. The central intervention in *Baba in Wonderland* is a remapping of the surface labels attached to `rule_property` tokens. This remapping changes only the words that appear in serialized states and transition prompts. It does not alter the underlying simulator dynamics, the action space, or the mechanics by which rules are parsed and executed. As a result, two environments that differ only by this remapping induce the same transition law even though they present different textual cues to the LLM. The concrete remapping used in our experiments is listed in Table 7.

The purpose of this intervention is to break the direct alignment between pretrained lexical priors and the true behavioral meaning of rule-property tokens. A model can no longer rely on the everyday meaning of an observed label to guess that it should behave like YOU, WIN, or STOP. Instead, it must infer the role of each remapped token from transition evidence and from how that token participates in active rule sentences over time. The agent is given no natural-language explanation of the remapping and no auxiliary dictionary that translates remapped labels back to their original meanings.

This design preserves the combinatorial and state-dependent structure of *Baba Is You* while introducing prior misalignment directly into the observable rule vocabulary. Success in *Baba in Wonderland* therefore provides stronger evidence that the learned executable world model is grounded in interaction evidence rather than in semantically aligned textual shortcuts.

A.2 Prompt Templates

Figure 4 shows the inductive program-update prompt template used by the Programmer. The template provides the current program, the target explanatory failure, and broken examples selected from hypothesis classes.

Inductive Program Update Prompts

````
You are preparing a new JSON patch for a Python world model program.

Your task:
- Base your reasoning on the structured state representation and the observed transition differences.
- Use the target failures as the primary objective.
- Edit only the accepted baseline program shown below.
- Prefer repairs that improve structural simplicity by reusing existing logic and minimizing special-case conditions.
- Keep `predict_next_state` as the top-level orchestrator of the transition pipeline.
- Revise, replace, consolidate, split, or delete functions according to the general rule they should represent.

$additional_instructions_block
$sandbox_runtime_block
Accepted baseline program to patch now:
- This is the only program you may edit.
```python
$current_source
```

Accepted baseline target failures to solve with the next patch:
- These are the direct failures the next patch must fix.
- Any field named accepted_baseline_predicted_next_state is the accepted baseline output.
$failed_text

$previous_patch_context_block$regression_witness_block$previous_smoke_errors_block$current_errors_block
Generate exactly one patch that edits the accepted baseline program.

Patch schema (STRICT):
- Return one JSON object in this exact shape:
{
  "edits": [
    {"op":"replace"|"add","name":"function_name","code":"def ..."}
  ],
  "delete_functions": ["optional_name"]
}

Rules:
- Edit only the accepted baseline program shown above.
- Output exactly one JSON object matching the patch schema.
- Do not use import statements, comments, or docstrings.
- Keep function names stable, and ensure `predict_next_state(state, action)` exists and returns a dict.
- Avoid runtime errors using explicit type and bounds checks.
- Do not use try/except, with, class, lambda, global, nonlocal, async/await, or yield.

Patch semantics:
- Each edit must provide one complete Python function definition in `code`, not a body fragment.
- Multiple edits apply in order.
- `replace`: replace one existing function in the accepted baseline program.
  Use this for `predict_next_state` and for any function that already exists, including nested functions.
- `add`: add one new top-level function whose name does not exist when this edit is applied.
  Its `code` must be an unindented top-level `def ...`; never use it for existing or nested functions.
- `delete_functions`: delete existing functions by name before applying `edits`.
  `predict_next_state` cannot be deleted; missing names are ignored, top-level matches are deleted before nested ones,
and ambiguous same-level matches are rejected.
````

Figure 4: Inductive program update prompt template.

Closed-loop procedure. Alice first finds a transition that its current program gets wrong. If a candidate fixes it but breaks previously explained examples, Alice uses those broken examples to guide the next update and organize exploration.

Algorithm 1 Alice: online executable world-model learning

Require: Interaction budget H, initial program P_0, dataset D, classes $\mathcal{C} \leftarrow \emptyset$

while interaction and LLM-call budgets remain **do**
  if initial root classes have not formed **then**
    Select frontier candidates by BFS
  else
    Update the embedding on active classes when due
    Select candidates with highest $\lambda_h r_h + \lambda_{\mathcal{C}} r_{\mathcal{C}}$
  end if
  Execute selected actions and add observed transitions to D
  for each transition τ not explained by P_i **do**
    $R \leftarrow \emptyset$
    for up to 15 update attempts, within the global call budget **do**
      $Q \leftarrow \mathrm{LLM}(P_i, \tau, R)$
      if Q explains τ and preserves every transition in $E(P_i; D)$ **then**
        Accept Q; update root classes and route new transitions; **break**
      else if Q explains τ but breaks previously explained transitions **then**
        Split partially affected classes into preserved and broken subsets
        $R \leftarrow$ one broken example from each of up to three affected classes
      end if
    end for
    Stop if the update fails; apply the remaining stopping rules in Appendix A.7
  end for
end while

A.3 Five-Run Analyses and Additional Evaluation

Evaluation splits and reporting. All main comparisons and ablations report means and standard deviations over five independent runs. The independent unit is a complete run, including LLM updates, exploration, and termination, rather than an individual transition. Online learning collects evidence on the 32 regular levels; its primary evaluation uses regular-level solution transitions. Offline learning receives regular-level solution transitions and is evaluated on the eight extra levels. These protocols differ in both evidence source and evaluation set, so their accuracies do not establish that offline or online learning is intrinsically superior. The offline setting also removes the exploration burden by supplying solution transitions directly.

Class-coverage coefficient. We swept $\lambda_{\mathcal{C}}$ with $\lambda_h = 1$ (Table 8). The five-run means are high over $\lambda_{\mathcal{C}} \in [0.2, 0.5]$; we use 0.35, which is near the center of this region and has the highest mean. This coefficient is fixed across the main online Alice experiments, including ALFWorld; the offline protocol has no Explorer. The sweep describes sensitivity within the evaluated configurations; it does not establish that every positive coefficient improves performance.

Table 8: Class-coverage sensitivity on regular-level solution transitions in Wonderland, with $\lambda_h = 1$. Five-run mean $\pm$ standard deviation.

Setting	All Acc. ↑	Balanced Acc. ↑	Calls
w/o $r_{\mathcal{C}}$	0.929 ± 0.050	0.890 ± 0.062	87.0 ± 20.5
$\lambda_{\mathcal{C}} = 0.05$	0.928 ± 0.050	0.890 ± 0.058	82.0 ± 18.9
$\lambda_{\mathcal{C}} = 0.10$	0.953 ± 0.045	0.923 ± 0.063	96.2 ± 8.5
$\lambda_{\mathcal{C}} = 0.20$	0.953 ± 0.045	0.926 ± 0.060	100.0 ± 0.0
$\lambda_{\mathcal{C}} = 0.35$ (**Alice**)	0.965 ± 0.054	0.951 ± 0.068	94.0 ± 10.4
$\lambda_{\mathcal{C}} = 0.50$	0.961 ± 0.057	0.937 ± 0.091	87.8 ± 13.8

Extra-level evaluation of online programs. Table 9 evaluates the final programs from the same five online runs on extra-level solution transitions, without further program updates. All learning interactions come from regular levels. This evaluates generalization beyond the levels used for interaction. The main online comparison and the statistical tests use regular-level solution transitions for all methods.

Table 9: Extra-level solution-transition accuracy of the online Alice runs. All rows use Wonderland unless stated otherwise. Calls are those of the regular-level learning runs. Five-run mean $\pm$ standard deviation.

Setting	All Acc. ↑	Balanced Acc. ↑	Calls
Alice ($\lambda_{\mathcal{C}} = 0.35$)	0.957 ± 0.056	0.945 ± 0.067	94.0 ± 10.4
Single-Class	0.877 ± 0.065	0.829 ± 0.081	84.8 ± 20.9
Root-Class	0.937 ± 0.062	0.911 ± 0.084	80.0 ± 12.8
w/o r_h	0.924 ± 0.046	0.900 ± 0.056	83.0 ± 13.5
w/o $r_{\mathcal{C}}$	0.902 ± 0.071	0.869 ± 0.091	87.0 ± 20.5
BFS	0.861 ± 0.104	0.811 ± 0.136	86.8 ± 18.6
$\lambda_{\mathcal{C}} = 0.05$	0.914 ± 0.070	0.886 ± 0.084	82.0 ± 18.9
$\lambda_{\mathcal{C}} = 0.10$	0.926 ± 0.073	0.898 ± 0.097	96.2 ± 8.5
$\lambda_{\mathcal{C}} = 0.20$	0.950 ± 0.067	0.930 ± 0.088	100.0 ± 0.0
$\lambda_{\mathcal{C}} = 0.50$	0.943 ± 0.116	0.922 ± 0.153	87.8 ± 13.8
Alice (Default World)	0.999 ± 0.002	0.999 ± 0.002	19.8 ± 6.6

Statistical tests for the ablations. All tests use the regular-level evaluation and the run as the independent unit. For each Alice–variant comparison, we enumerate all $\binom{10}{5} = 252$ assignments in an exact two-sided permutation test, applying Holm correction across the five comparisons separately for each metric. Exact Mann–Whitney tests give the same significance conclusions after correction. No individual comparison is significant after Holm correction at $\alpha = 0.05$ (Table 10). The smallest attainable two-sided permutation p-value with five runs per group is $2/252 \approx 0.008$; the small run count limits resolution. Differences below are calculated before rounding the reported means.

Table 10: Pairwise Alice–variant differences and exact two-sided permutation p-values. Holm correction is over five comparisons per metric.

	All Acc.		Balanced Acc.	
Variant	Δ	p (raw / Holm)	Δ	p (raw / Holm)
Single-Class	$+0.075$	$.056/.278$	$+0.122$	$.024/.119$
Root-Class	$+0.025$	$.460/.919$	$+0.044$	$.357/.714$
w/o r_h	$+0.012$	$.516/.919$	$+0.026$	$.492/.714$
w/o $r_{\mathcal{C}}$	$+0.037$	$.230/.690$	$+0.061$	$.183/.549$
BFS	$+0.064$	$.071/.283$	$+0.096$	$.079/.317$

We also test three orderings induced by progressively adding class structure or exploration signals. Each exact one-sided Jonckheere–Terpstra permutation trend test enumerates all $15!/(5!)^3 = 756{,}756$ assignments of the fifteen runs, with Holm correction across the three orderings separately for each metric. All three trends remain significant on both metrics (Table 11). This supports an overall monotonic trend, not a significant improvement for every adjacent pair or every individual component. In particular, w/o r_h and w/o $r_{\mathcal{C}}$ form two separate paths from BFS to Alice; they are not ordered relative to one another.

Table 11: Exact one-sided permutation trend tests; entries are raw / Holm-adjusted p-values.

Ordered hypothesis	All Acc.	Balanced Acc.
Single-Class $<$ **Root-Class** $<$ **Alice**	$.006/.019$	$.004/.012$
BFS $<$ **w/o** $r_{\mathcal{C}}$ $<$ **Alice**	$.010/.019$	$.028/.043$
BFS $<$ **w/o** r_h $<$ **Alice**	$.009/.019$	$.021/.043$

Same-call-cap WorldCoder diagnostic. Under the strict protocol, an online run terminates after exhausting the 15-call budget on a single failed update. To separate this stopping rule from the global call cap, the diagnostic retains WorldCoder's last accepted program after such a failure and continues collecting evidence and attempting later updates up to 100 calls. Any unfinished update at the cap is discarded. This changes the continuation rule; it matches the global cap, not the realized number of calls. Table 12 reports five-run results. WorldCoder improves from 0.542 to 0.819 All Acc. and from 0.260 to 0.658 Balanced Acc., showing that early termination accounts for part of the strict-protocol gap. Alice still has a 0.293 higher mean Balanced Acc. under the same cap.

Table 12: Same-100-call-cap diagnostic on regular-level solution transitions in Wonderland. Five-run mean $\pm$ standard deviation.

Method	All Acc. $\uparrow$	Balanced Acc. $\uparrow$	Calls
Alice	0.965 ± 0.054	0.951 ± 0.068	94.0 ± 10.4
WorldCoder, continued	0.819 ± 0.052	0.658 ± 0.131	100.0 ± 0.0

A.4 Information-Theoretic View of Frontier Scoring

This section gives an information-theoretic view of the frontier score used by the Explorer. The score is designed to make frontier coverage program-relative: the Explorer should cover frontier candidates in a way that is informative about the hypothesis classes induced by the accepted program-update sequence. Let $C_{\tau_x} \in \mathcal{C}$ denote the hypothesis class assigned to the induced transition τ_x after executing frontier candidate x. This motivates the objective

$$\max_{\pi} H_\pi(x) + I_\pi(C_{\tau_x}; x).$$

The entropy term $H_\pi(x)$ expands coverage over frontier candidates, but by itself it may only increase surface variation among state-action pairs. The mutual-information term $I_\pi(C_{\tau_x}; x)$ makes this coverage class-informative: when x carries information about C_{τ_x}, increasing coverage over x is aligned with increasing coverage over hypothesis-class evidence.

This form follows two ideas from reward-free exploration and skill discovery. APT motivates entropy-based coverage as a reward-free exploration objective [21]. DIAYN and CIC motivate mutual information as a way to make a diversity variable recoverable from generated experience or learned representations [7, 18]. Alice changes the role of this variable: C_{τ_x} is not a skill, option, or behavior variable to be executed, but a hypothesis class induced by accepted program updates. The purpose is therefore not to learn a reusable skill repertoire, but to make frontier coverage relative to the current executable program.

Because $x = (s, a)$ is a structured state-action object, we formulate the criterion in a learned representation space, with $h = h_\theta(x)$. The representation-level objective is

$$\max_{\pi,\theta} H_\pi(h) + I_\pi(C_{\tau_x}; h).$$

The role of h_θ is to preserve hypothesis-class distinctions while providing a space in which coverage can be estimated.

Using the mutual-information identity,

$$I_\pi(C_{\tau_x}; h) = H_\pi(C_{\tau_x}) - H_\pi(C_{\tau_x} \mid h),$$

we obtain

$$J_{\text{explore}} = H_\pi(h) - H_\pi(C_{\tau_x} \mid h) + H_\pi(C_{\tau_x}).$$

Maximizing this objective imposes three pressures. The first term, $H_\pi(h)$, encourages coverage in the learned embedding space. The second term, $-H_\pi(C_{\tau_x} \mid h)$, makes the embedding class-informative. The final term, $H_\pi(C_{\tau_x})$, encourages coverage across hypothesis classes, preventing the policy from satisfying the objective by expanding variation inside only one or a few classes.

The frontier score in Section 4.2 is a computable approximation of this criterion. The embedding-novelty score $r_h(x)$ approximates the $H_\pi(h)$ term, the active-class contrastive objective supports the $-H_\pi(C_{\tau_x} \mid h)$ term, and the expected class-coverage score $r_{\mathcal{C}}(x)$ approximates the $H_\pi(C_{\tau_x})$ term. Thus,

$$S_{\text{frontier}}(x) = \lambda_h r_h(x) + \lambda_{\mathcal{C}} r_{\mathcal{C}}(x)$$

can be viewed as a practical frontier-scoring approximation to the entropy–mutual-information objective.

Table 13: Hypothesis-class alignment for Alice. Purity is measured from heuristic transition classes $\mathcal{C}_H$ to Alice's learned classes $\mathcal{C}$ on human-play solution transitions.

Setting	Evaluation Set	Purity ↑	$\#\mathcal{C}_H$	$\#\mathcal{C}$
Default World	Regular levels	0.978	595	23
Default World	Extra levels	0.955	293	14
Baba in Wonderland	Regular levels	0.793	595	66
Baba in Wonderland	Extra levels	0.817	293	36

A.5 Additional Analyses

This subsection complements the main experiments with analyses of learning behavior, class alignment, embedding structure, transition coverage, update-evidence hyperparameters, and LLM backbones. The learning curves average the five runs used in the main ablations; the other analyses use a single run per configuration.

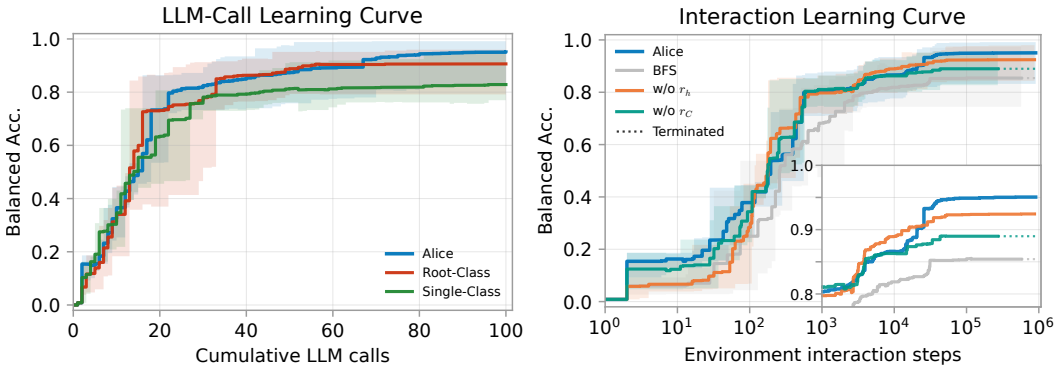

Figure 5: Learning curves averaged over five runs; shading spans the minimum and maximum over runs, and a stopped run keeps its final accuracy. Left: hypothesis-class variants over cumulative LLM calls. Right: exploration variants over interaction steps; a dotted line starts once all five runs of a method have stopped.

Learning curves. Figure 5 (left) shows how the hypothesis-class variants fall behind. Two of the five Single-Class runs exhaust the 15-call budget on a single update and stop at 59 and 65 calls. Without classes, the three broken examples in the update context are drawn at random from all broken transitions, so they can repeat one preservation constraint while missing the others that a candidate breaks. The other three Single-Class runs accept 24–28 additional updates per run after call 40, compared with 17–30 per run for Alice, but improve Balanced Acc. by only 0.057 on average, compared with 0.097 for Alice. With a single class, the class-coverage term is zero, so the Explorer cannot direct evidence toward weakly supported parts of the program. Root-Class tracks Alice up to about 66 calls, and their mean curves cross several times. Early root classes still hold few distinct constraints, so one randomly drawn broken example per root class is often as informative as one per refined class. As accepted updates add transitions to these root classes, each comes to mix several constraints, and a random draw more often misses the one that a candidate breaks. Three of the five Root-Class runs then exhaust the per-update budget, against none for Alice, and Alice ends above Root-Class for every seed.

Figure 5 (right) shows a similar late separation for exploration. All frontier-scoring explorers reach about 0.80 Balanced Acc. by 10^3 steps, against 0.68 for BFS. The two score ablations stay close to Alice up to about 2.5×10^4 steps. Between 10^4 and 10^5 steps, Alice gains 0.083 in Balanced Acc., compared with 0.027–0.036 for the ablations. The score-ablation runs also stop earlier: the median run ends after 2.6×10^4 steps without r_h and 3.4×10^4 steps without r_C, against 5.2×10^5 for Alice. Two or three of the five runs in each ablation, including BFS, exhaust the per-update budget, against none for Alice. Once common transitions are explained, the remaining errors involve rare dynamics, and reaching them appears to require both novelty and coverage of weakly supported classes.

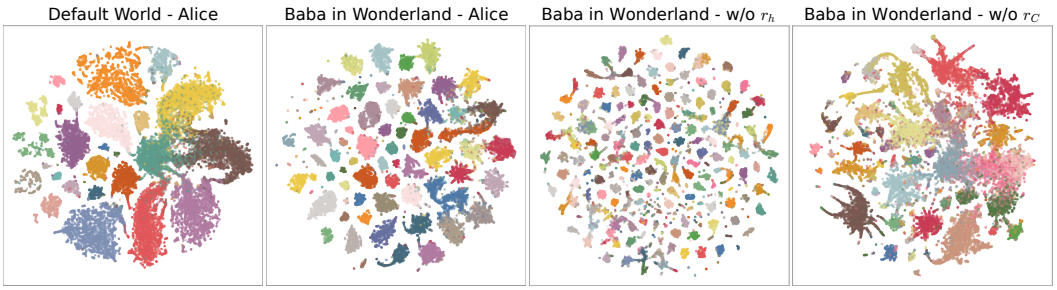

Figure 6: t-SNE visualization of the Explorer's learned state-action embedding space. Points are explained transitions projected from the 384-dimensional embedding space, and colors denote assigned hypothesis classes. From left to right: Alice in *Default World*, Alice in *Baba in Wonderland*, **w/o** r_h in *Baba in Wonderland*, and **w/o** r_C in *Baba in Wonderland*.

Table 14: Evaluation on the held-out extra-level coverage transition dataset. All Acc. is measured on the full coverage transition dataset; Balanced Acc. is measured on the corresponding class-reduced coverage transition set.

	Default World		Baba in Wonderland	
Method	All Acc. ↑	Balanced Acc. ↑	All Acc. ↑	Balanced Acc. ↑
WorldCoder	0.984	0.894	0.749	0.029
Alice	0.999	0.996	0.997	0.995

Hypothesis-class alignment. We further test whether Alice's learned hypothesis classes remain tied to transition structure when semantic shortcuts are removed. For this diagnostic, we evaluate Alice's classes $\mathcal{C}$ on the solution transition dataset and compare them with an evaluation-only heuristic partition $\mathcal{C}_H$, which groups transitions by action and canonical state-difference signature and is never used during learning. We report micro-averaged one-way purity from $\mathcal{C}_H$ to $\mathcal{C}$: for each heuristic class, we assign its evaluated transitions to their majority Alice class and aggregate the majority counts across heuristic classes. The reported $\#\mathcal{C}$ is therefore the number of Alice classes represented among the evaluated solution transitions, not the total number of classes discovered in the broader exploration archive. As shown in Table 13, purity remains substantial in both *Default World* and *Baba in Wonderland*, despite a drop under the misleading Wonderland labels. This indicates that transitions with the same behavioral effect remain largely grouped under Alice's learned partition, even when the surface rule-property labels no longer provide reliable semantic cues.

Embedding space t-SNE visualization. Figure 6 provides a qualitative view of the learned state-action embeddings. Each panel samples 50,000 explained transitions, projects their 384-dimensional embeddings with t-SNE, and colors points by assigned class. Alice forms class-structured embeddings in both label settings. Removing either exploration term changes the distribution of visited classes: without r_h, the view contains smaller fragmented clusters; without r_C, samples concentrate in a few larger classes. These views complement the exploration ablations by illustrating the organization of collected evidence; distances in the projection do not directly measure state-space coverage.

Coverage transition dataset evaluation. The main experiments evaluate on human-play solution transitions, which concentrate on behavior encountered along successful solution trajectories. As a complementary stress test, we evaluate the online runs trained on regular levels on the coverage transition dataset collected by BFS over each held-out extra level. All Acc. uses the full coverage transition dataset, while Balanced Acc. uses the corresponding class-reduced coverage transition set to discount repeated local effects. Table 14 shows that WorldCoder remains much more reliable in *Default World* than in *Baba in Wonderland*; the gap is especially large under Balanced Acc., indicating that lexical shortcuts can preserve common coverage transitions while failing to recover diverse transition effects under prior misalignment. By contrast, Alice maintains high accuracy on the coverage transition dataset in *Baba in Wonderland*, showing that its learned executable model transfers beyond human-play solution trajectories.

Table 15: Offline ablation of Alice's program-update evidence hyperparameters in *Baba in Wonderland*. The default Alice setting is $n = 3, m = 1$.

Evidence Setting	All Acc. ↑	Balanced Acc. ↑	Calls
$n = 1, m = 1$	0.722	0.621	51.0
$n = 1, m = 3$	0.857	0.809	100.0
$n = 3, m = 1$ (**Alice**)	0.956	0.932	100.0
$n = 3, m = 3$	0.964	0.911	100.0

Table 16: Alice with different LLM backbones in *Baba in Wonderland*. Results are online regular-level human-play transition accuracy under the same exploration and update protocol.

Backbone	All Acc. ↑	Balanced Acc. ↑	Calls
GPT-5.4-mini	0.865	0.713	41.0
Gemini 3.0 Flash	0.958	0.903	56.0
GPT-5.4	0.982	0.973	100.0
Gemini 3.1 Pro	0.993	0.992	65.0

Program-update evidence hyperparameters. Alice constructs the LLM update context by selecting up to n affected hypothesis classes and sampling up to m broken examples from each selected class. The main experiments use $n = 3$ and $m = 1$. We ablate these values in the offline fixed-dataset setting, where all methods train on fixed regular-level human-play transitions and are evaluated on held-out extra-level human-play transitions under a 100-call cap. Table 15 shows that performance is sensitive to how selected broken examples are packaged. The default $n = 3, m = 1$ setting gives the strongest Balanced Acc., while $n = 3, m = 3$ slightly improves All Acc. at the cost of lower Balanced Acc. The $n = 1$ variants are substantially weaker, indicating that too little class diversity in the program update context can miss rare preservation constraints. We therefore treat these results as a sensitivity analysis: class-selected broken examples are useful, but larger update contexts are not automatically better under a fixed LLM-call budget.

LLM backbone variation. We evaluate Alice with different LLM backbones while keeping the prompting format, program-update loop, exploration strategy, and evaluation protocol fixed. Table 16 reports online performance on regular-level human-play transitions in *Baba in Wonderland*. The runs show different update dynamics across backbones. In our qualitative inspection, Gemini 3.1 Pro quickly produced a broad program that achieved high accuracy, but later had difficulty revising that program for a few special cases and terminated before using the full call budget. GPT-5.4 instead tended to make more incremental program revisions and continued until the full 100-call budget was exhausted. The smaller-model runs terminated earlier and reached lower accuracy, especially under Balanced Acc. These results suggest that LLM backbones can differ not only in final accuracy, but also in how they revise executable programs; Alice's class-structured update and exploration loop nevertheless remains usable across the tested backbones.

A.6 Qualitative Examples

For readability, the visualizations in this subsection render states with the original *Baba Is You* sprites, although the evaluated transitions and learned programs come from *Baba in Wonderland*; only the textual rule-property labels are remapped in the agent's observations.

Success cases. Figure 7 shows an evaluation transition on which Alice predicts the exact next state while all three baselines fail. In the underlying game semantics, `BABA IS YOU`, `ROCK IS PUSH`, and `JELLY IS PUSH` are active; in *Baba in Wonderland*, these appear as `BABA IS STRANGE`, `ROCK IS GROW`, and `JELLY IS GROW`. When `BABA` moves right, the correct transition is a two-object push chain: `JELLY` moves into the empty cell, `ROCK` moves into the `JELLY`'s previous cell, and `BABA` moves into the `ROCK`'s previous cell. Alice's learned program succeeds because it infers that the surface label `GROW` plays the role of `PUSH` and applies a recursive chain movement routine before moving the controlled object. The selected WorldCoder program moves controlled objects but does not model this push-chain effect, while the GIF-MCTS and CWM programs contain generic push logic but do

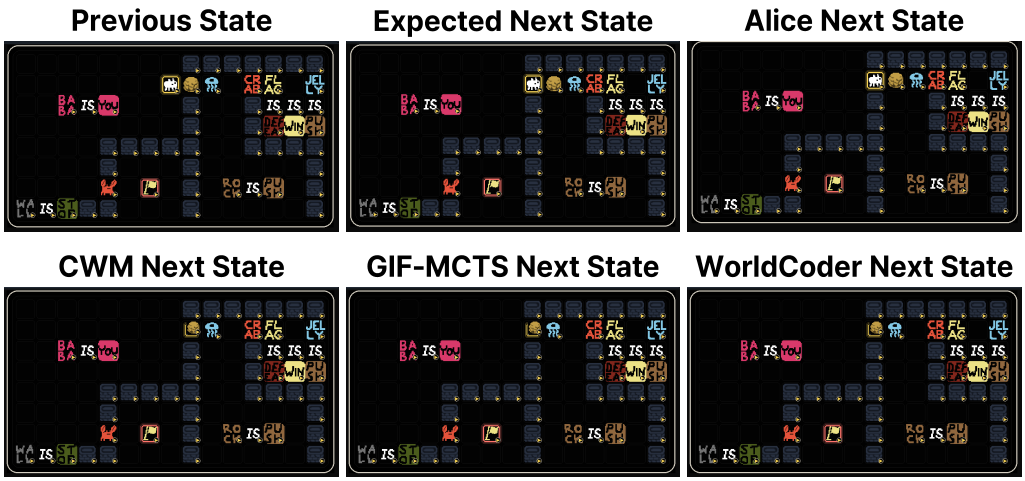

Figure 7: Success Case 1 from *Baba in Wonderland*: a push chain under misleading rule-property labels. Each method panel shows the previous state, expected next state, and the method's predicted next state.

not identify the Wonderland GROW property as PUSH in this configuration. All three baselines therefore leave the ROCK–JELLY chain behind, whereas Alice recovers the full transition from evidence despite the misleading property label.

Figure 8 shows a different kind of success case involving simultaneous movement and blocking. Here BABA IS YOU and ROCK IS YOU make both BABA and ROCK controlled objects, while HEDGE IS STOP makes HEDGE blocking objects; in *Baba in Wonderland*, these rules appear as BABA IS STRANGE, ROCK IS STRANGE, and HEDGE IS EAT. After the upward action, the lower controlled objects move up, but the upper ROCK is immediately below a HEDGE and should therefore remain in place while updating its facing direction. Alice's learned program handles this mixed outcome: it moves each controlled object in the correct order, checks the blocking constraint for the ROCK adjacent to the HEDGE, and preserves the blocked ROCK rather than deleting it or moving it into the HEDGE cell. The baselines miss this interaction in different ways: WorldCoder lacks the corresponding stop-blocking rule, GIF-MCTS omits the blocked ROCK, and CWM moves the ROCK into the blocked cell. This case illustrates why balanced transition accuracy is diagnostic: the difficult part is not moving a controlled object in isolation, but preserving the exact simulator semantics when several controlled objects interact with blocking structure at the same time.

Failure cases. A representative failure case is shown in the top row of Figure 9. In this configuration, WALL IS SHUT, STAR IS OPEN, and STAR IS PUSH are active; in *Baba in Wonderland*, these appear as WALL IS LATE, STAR IS MAD, and STAR IS GROW. When BABA pushes the KEY text block, the pushed block in turn displaces the STAR into the WALL. Under the simulator dynamics, the resulting OPEN–SHUT interaction should remove both the STAR and the WALL, even though the STAR was moved indirectly rather than directly by a YOU object. Alice's learned program fails to predict this cancellation and leaves the objects in place. The error comes from an over-simplified collision routine in the induced program: because many important interactions in the data are triggered by agent-controlled movement, the learned code checks object–object interaction effects primarily for motion initiated by YOU-controlled objects. In this case, however, the relevant overlap is created by a STAR that is displaced by another pushed object, not by BABA itself. As a result, the learned program misses the OPEN–SHUT resolution that the simulator applies, illustrating how Alice can fit a compact but overly narrow causal account of collision dynamics.

A second failure case is shown in the middle row of Figure 9. Here a noun-to-noun transformation rule of the form PILLAR IS BABA becomes active, and the learned program correctly converts the PILLAR objects into BABA. However, one of the transformed PILLAR objects occupies a cell already overlapping a STAR object with the DEFEAT property. Under the simulator dynamics, the newly

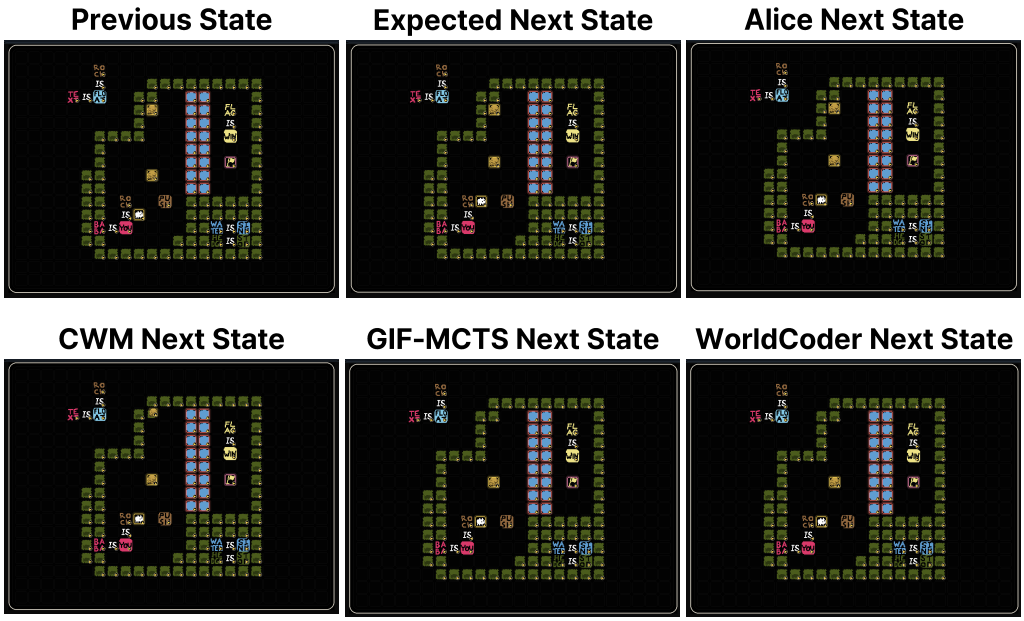

Figure 8: Success Case 2 from *Baba in Wonderland*: simultaneous movement of multiple YOU objects with one controlled object blocked by a stop object. Each method panel shows the previous state, expected next state, and the method's predicted next state.

created BABA should then be removed immediately, because BABA is a YOU object and thus dies on overlap with DEFEAT. Alice's learned program performs the noun-to-noun conversion but does not resolve the resulting world-object interaction afterward, so the transformed BABA incorrectly remains in the predicted next state. This error reflects a highly specific blind spot in the induced program: it models object transformation and object–object collision as partially separate cases, but does not apply collision resolution after a fresh noun-to-noun conversion. Because such transitions are rare in the learning data, the resulting program never acquires the more complete rule that conversion can immediately trigger post-transform interaction effects.

A third failure case is shown in the bottom row of Figure 9. In this configuration, ROCK IS ROCK is active when a competing noun-to-noun rule ROCK IS FLAG is formed. Under the simulator dynamics, the self-identity rule takes precedence: once A IS A is active, it blocks other noun-to-noun rewrites for that object, so the existing ROCK objects should remain ROCK rather than transforming into FLAG. Alice's learned program instead applies ROCK IS FLAG and converts the ROCK objects into FLAG, producing an incorrect next state. This failure reflects a missing rule-ordering constraint in the induced program. From observed transitions alone, it is difficult to infer that self-identity rules are not merely redundant transformations but priority-setting constraints that suppress competing noun conversions. Because this precedence relation appears only in relatively specific configurations, the learned program captures the surface conversion pattern but not the underlying execution order that makes ROCK IS ROCK dominate ROCK IS FLAG.

Example update trace. Figure 10 makes the class-refinement mechanism concrete through one actual update trace from Alice in *Baba in Wonderland*. Alice reaches a transition in the level *Burglary* where BABA IS STRANGE (YOU) is active and the action is right. The correct next state moves BABA one cell to the right, but it also moves the adjacent rule-text chain IS LATE one cell to the right. Here LATE is the Wonderland surface label for SHUT. The current program explains ordinary controlled movement, but it does not yet model this case where a controlled object pushes rule text. The transition therefore becomes an explanatory failure.

The first candidate update fixes this target: it moves both BABA and the adjacent text chain. However, the executable acceptance check also tests whether the candidate preserves transitions that the previous

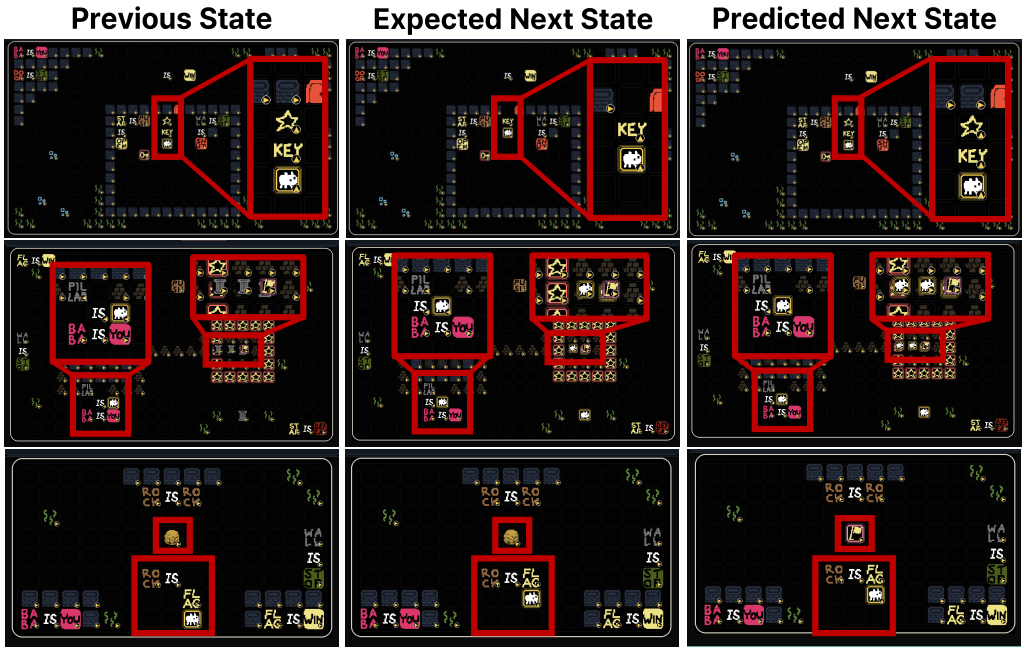

Figure 9: Failure case studies from *Baba in Wonderland*.

program already explained. The candidate fails this check. In a previously explained transition from the level *Off Limits*, `BABA IS STRANGE` (`YOU`) is active and the action is `left`. The correct next state moves only `BABA` one cell left. The candidate instead overgeneralizes the new push rule and also moves nearby non-controlled objects, including a `TILE` and a `WALL`. Thus, the candidate has learned a useful part of the target behavior, but it has confused "pushing adjacent rule text" with a broader rule that moves unrelated neighboring objects.

Alice uses this rejection to refine its hypothesis classes. The affected class is split into transitions that the rejected candidate still preserves and transitions that it breaks. The preserved subset contains cases compatible with the candidate's new behavior; the lost subset contains broken examples showing where that behavior overgeneralizes. Alice samples a representative lost transition, such as the *Off Limits* case above, and includes it in the next update prompt as a selected broken example. The next candidate is therefore asked to explain the original *Burglary* failure while also preserving the *Off Limits* broken example. That retry is accepted with no preservation losses. The trace in Figure 10 shows why rejected updates are informative in Alice: they reveal which transitions were incorrectly conflated, split the corresponding hypothesis class, and determine which broken examples the next LLM update sees.

A.7 Experimental Details

Regular and extra levels. We use original *Baba Is You* levels drawn from Stages 1, 2, and 3 of the base game. Following the level taxonomy of the game, we distinguish *regular levels* from *extra levels*; this partition is defined over levels, not by splitting trajectories from the same level. The regular set contains 32 levels: all 8 Stage-1 levels, 13 regular Stage-2 levels, and 11 regular Stage-3 levels. The extra set contains 8 levels: 2 Stage-2 extra levels and 6 Stage-3 extra levels. The level display names in each set are listed in Table 17. Online experiments collect interaction data on regular levels and evaluate on human-play solution transitions from those same levels, whereas offline fixed-dataset experiments train on regular-level human-play transitions and evaluate on held-out extra-level human-play transitions.

Dataset construction. For each level in the regular and extra splits, we construct two transition datasets. The *solution transition dataset* is obtained from successful human solution trajectories

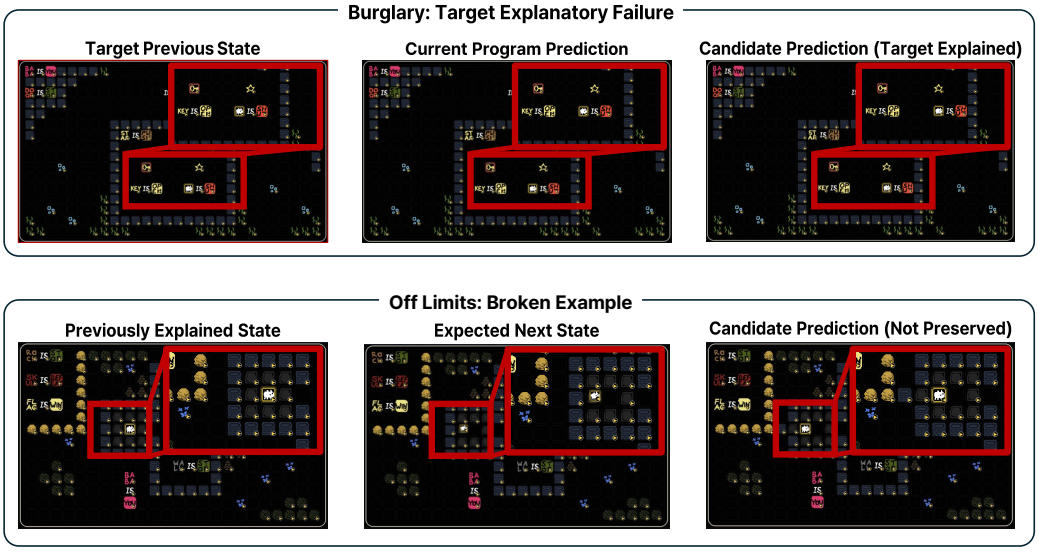

Figure 10: Actual rejected candidate update from Alice in *Baba in Wonderland*. Top: the candidate explains the target transition from *Burglary*, where BABA pushes adjacent rule text. Bottom: the same candidate fails to preserve a previously explained transition from *Off Limits*, exposing a broken example for the next update prompt.

Table 17: Original *Baba Is You* level display names used in the regular and extra level sets.

Stage	Regular Levels	Extra Levels
1	*Baba Is You*; *Where Do I Go?*; *Now What Is This?*; *Out Of Reach*; *Still Out Of Reach*; *Volcano*; *Off Limits*; *Grass Yard*	–
2	*Icy Waters*; *Turns*; *Affection*; *Pillar Yard*; *Brick Wall*; *Lock*; *Novice Locksmith*; *Locked In*; *Changeless*; *Two Doors*; *Jelly Throne*; *Crab Storage*; *Burglary*	*Submerged Ruins*; *Sunken Temple*
3	*Float*; *Warm River*; *Bridge Building*; *Bridge Building?*; *Victory Spring*; *Assembly Team*; *Catch The Thief!*; *Tiny Pond*; *Research Facility*; *Wireless Connection*; *Prison*	*Boiling River*; *. . . Bridges?*; *Tiny Isle*; *Dim Signal*; *Dungeon*; *Evaporating River*

and emphasizes the transitions that arise along task-completing behavior. The *coverage transition dataset* is obtained by running BFS on each level and recording canonical transitions until either the reachable frontier is exhausted or $100{,}000$ canonical transitions have been collected for that level. Both datasets are stored as transition archives paired with state archives. Within each level-specific dataset, we canonicalize states and remove duplicate transitions that share the same (s, a, s') triple. This ensures that repeated visits to the same local transition effect do not inflate the archived dataset and that the $100{,}000$-transition cap for coverage is applied to canonical transitions rather than raw observations. Aggregate dataset statistics are summarized in Table 18. The class-reduced subsets used for balanced accuracy are derived from these full transition datasets by the heuristic procedure described in Appendix A.9.

Evaluation protocol and cost measurement. After training, we evaluate the selected executable world model on the full and class-reduced variants of the solution transition dataset and coverage transition dataset for each split. All Acc. is measured on the full transition dataset, whereas Balanced Acc. is measured on the corresponding class-reduced subset derived by heuristic dynamics discovery. Our primary metric is one-step transition accuracy. For each archived transition (s, a, s'), a prediction

Table 18: Archived solution and coverage transition dataset statistics after canonical transition deduplication.

Split	Dataset	# Levels	# Transitions	# States
Regular Levels	Solution transition dataset	32	2,476	2,490
Regular Levels	Coverage transition dataset	32	3,200,000	1,254,186
Extra Levels	Solution transition dataset	8	1,126	1,126
Extra Levels	Coverage transition dataset	8	800,000	306,488

is counted as correct only if the program executes successfully on (s, a) and its predicted next state exactly matches the archived next state after canonicalization. Compile failures, contract violations, missing predictions, and runtime errors are all counted as incorrect predictions. We report regular-set and extra-set accuracy separately by aggregating the corresponding archived datasets over levels within each split. For LLM-based methods, we report cumulative training-time Calls, token composition, and relative weighted token cost; Calls alone is not a total-cost measure (Appendix A.10).

Compute resources. Experiments were run on a workstation with an Intel Xeon Gold 6226R CPU at 2.90GHz, 128GB RAM, and two NVIDIA GeForce RTX 3090 GPUs. The operating system was Ubuntu 20.04.6 LTS with Linux kernel 5.4.0-216-generic. Elapsed-time measurements for the five-run experiments are reported in Appendix A.10. External LLM latency affects wall-clock time; token usage and local computation are therefore reported separately.

Embedding architecture and optimization. In Baba, each object is a 58-dimensional token: a 4-dimensional one-hot type, a 45-dimensional one-hot word, a 5-dimensional one-hot direction, and four dimensions for normalized position and grid scale. We project tokens to 384 dimensions and encode their relations with four six-head Set Transformer blocks. Object order has no meaning, so we use no positional encoding. A query combining a learnable seed and a 384-dimensional action embedding pools the token set; an MLP and L2 normalization produce $h_\theta(s, a) \in \mathbb{R}^{384}$. The encoder has approximately 6.49 million trainable parameters. We use a learning rate of 10^{-4}, supervised-contrastive temperature 0.2, and two-GPU distributed data parallel training with global batch size 4,096. Every 4,096 added transitions trigger 30 optimization updates. Each active class is represented by up to 8 prototypes for frontier scoring. These object-token dimensions describe the Baba encoder, not ALFWorld's distinct symbolic state representation.

Update and stopping rules. Unless otherwise noted, the Baba online discovery runs use the same pipeline and differ only in the component being ablated. The ALFWorld interaction budget is specified in Section 5.1. Each run terminates when either the global LLM call budget of 100 is exhausted, a single program-update iteration uses 15 LLM calls without an accepted update, or no candidate update has been accepted for $300{,}000$ global interaction steps. Inductive program update uses a per-target retry cap of 15 rejected candidate updates. When a rejected candidate update causes previously explained transitions to become unexplained, we sample additional transition evidence by selecting up to 3 affected classes uniformly at random and then selecting one broken transition uniformly at random from each selected class. Candidate updates are accepted only if they resolve the target failure and introduce no compile errors, runtime errors, or explanation losses on the full set of previously explained transitions in D_t. For Explorer representation learning, we include a hypothesis class in the active class set only after it has at least 8 assigned transitions, and the learned state-action embedding has dimension 384.

Across LLM-based runs, we keep the prompting and decoding configuration fixed unless the experiment explicitly studies backbone variation. In particular, we use a medium reasoning or thinking setting together with a maximum output budget of $32{,}768$ tokens. In the backbone comparison, we change only the underlying LLM while keeping the update procedure, exploration strategy, prompts, and evaluation protocol unchanged.

A.8 ALFWorld Experiments

Setting and metric. The protocol in Section 5.1 includes all six household task types. Evaluation episodes are not used during learning, and the metric is exact next-state agreement; compile failures and runtime errors count as incorrect predictions. Alice uses the fixed coefficient $\lambda_{\mathcal{C}} = 0.35$. The main comparison trains both methods without a curriculum and reports means and standard deviations over five runs.

Comparison and curriculum. The curriculum comparison in Table 3 uses all six task types while preserving WorldCoder's original task order. It uses the same unseen-episode evaluation and call caps as the no-curriculum comparison.

WorldCoder's original ALFWorld results report average episode reward (task success) over three seeds with an MCTS planner and a BM25 heuristic. Its curriculum covers four of the six task types, and its learning-performance evaluation has no separate held-out split. Our six-task, unseen-episode transition evaluation therefore differs in metric, task coverage, and evaluation protocol. High planning success does not require exact predictions on every transition outside the successful plans, so the two reported accuracies should not be directly compared.

Token cost. Table 19 reports token usage under the same accounting convention as Appendix A.10. Alice's weighted cost is less than half that of WorldCoder in this setting, alongside higher prediction accuracy. This result is specific to ALFWorld and does not imply a cost advantage in Wonderland online learning. The curriculum diagnostic is not included in this cost table.

Table 19: ALFWorld token accounting. Token counts are in millions; weighted cost is measured in millions of input-token-equivalent units. Five-run mean $\pm$ standard deviation.

Quantity	**Alice**	**WorldCoder**
Input tokens	1.332 ± 0.467	2.861 ± 0.247
Reasoning tokens	0.010 ± 0.004	0.057 ± 0.009
Output tokens	0.029 ± 0.007	0.109 ± 0.029
Total tokens	1.371 ± 0.476	3.026 ± 0.258
Weighted cost	1.564 ± 0.525	3.856 ± 0.372

A.9 Heuristic Dynamics Discovery for Class-Reduced Subsets

Algorithm. The heuristic dynamics discovery procedure is applied to the archived transition datasets after collection. For each transition, we first reconstruct the source state and next state from the paired state archive. Each state is represented as a multiset of object tokens of the form $(\texttt{type}, \texttt{word}, x, y, \texttt{direction})$, together with the grid size and termination flags. We then compute a canonical state-difference signature that summarizes how the object multiset changes across the transition.

This signature records three kinds of object-level effects: moved objects, removed objects, and added objects. For moved objects, we match objects by `type`, `word`, and `direction`, and record only the displacement $(\Delta x, \Delta y)$ together with the object identity and multiplicity. For objects that disappear or appear without a matched counterpart, we record removal or addition counts by object identity. The signature also records changes in grid size, termination, or truncation when they occur.

Two transitions are assigned to the same heuristic class if and only if they have the same action and the same canonical state-difference signature. As a result, transitions that induce the same local dynamical effect but occur at different absolute positions or in different levels are collapsed into the same heuristic class. This heuristic is not intended to recover the true latent dynamics of the environment. Rather, it provides a simple way to remove large amounts of redundant transition evidence while preserving diversity over distinct observed transition effects.

Subset construction. Given the heuristic class assignments above, we construct a class-reduced subset by selecting one representative transition from each heuristic class. For each heuristic class, we select one representative transition by uniform random sampling with a fixed seed. The resulting subset therefore contains exactly one transition per heuristic class.

We apply this procedure separately to the solution transition dataset and the coverage transition dataset for each split. The resulting class-reduced subsets are used to compute Balanced Acc. for the solution transition dataset and coverage transition dataset. In our archived datasets, this reduces the train/test solution transition sets from $2{,}476/1{,}126$ transitions to $595/293$ representatives and the train/test coverage transition sets from $3{,}200{,}000/800{,}000$ transitions to $7{,}232/2{,}073$ representatives. These subsets are not intended to be harder benchmarks than the full datasets. Rather, they provide a complementary view that discounts repeated instances of the same local transition effect and emphasizes coverage over distinct heuristic dynamics classes.

A.10 Computational Cost and Scaling

API accounting. Calls alone do not measure total API cost. We record input, reasoning, and output tokens separately for each run. Under the pricing ratio used in these experiments, reasoning and output tokens each cost six times as much as input tokens. We therefore compute $\mathrm{Cost} = \mathrm{Input} + 6(\mathrm{Reasoning} + \mathrm{Output})$ per run and then report the mean and standard deviation. This is a relative token-price measure, not a dollar amount or a measure of local compute. Totals and costs are computed before rounding.

Table 20: Wonderland token accounting. Token counts are in millions; weighted cost is measured in millions of input-token-equivalent units. Five-run mean $\pm$ standard deviation.

Setting	Method	Input	Reasoning	Output
Online	**Alice**	2.867 ± 0.531	0.096 ± 0.015	0.097 ± 0.026
Online	**WorldCoder**	1.335 ± 0.778	0.053 ± 0.024	0.096 ± 0.015
Offline	**Alice**	3.020 ± 0.429	0.115 ± 0.019	0.132 ± 0.073
Offline	**GIF-MCTS**	2.144 ± 0.145	0.070 ± 0.046	0.230 ± 0.069
Offline	**CWM**	2.289 ± 0.160	0.167 ± 0.013	0.245 ± 0.032

Setting	Method	Total tokens	Weighted cost
Online	**Alice**	3.060 ± 0.555	4.025 ± 0.689
Online	**WorldCoder**	1.485 ± 0.764	2.232 ± 0.707
Offline	**Alice**	3.267 ± 0.497	4.500 ± 0.867
Offline	**GIF-MCTS**	2.444 ± 0.126	3.945 ± 0.079
Offline	**CWM**	2.701 ± 0.187	4.763 ± 0.319

In Wonderland online learning, Alice incurs a higher weighted token cost than WorldCoder while achieving higher accuracy. WorldCoder's lower expenditure is real, although it partly reflects termination at 57.6 calls on average. In the offline comparison, Alice's weighted cost lies between those of GIF-MCTS and CWM. Thus, lower call counts should not be read as a general total-cost advantage.

Embedding overhead. Table 21 separates frontier scoring, prototype refresh, and embedding training in the five online runs. Together these stages average 1.26 ± 0.93 hours, or $21.33 \pm 10.66\%$ of total wall-clock time. These are stage elapsed times, not GPU-hours. Percentages are computed within each run before averaging, so a ratio of the reported means need not equal the mean percentage. We do not infer a baseline wall-clock advantage because external LLM latency affects total elapsed time.

Table 21: Embedding-related elapsed time and share of total wall-clock time in online Wonderland runs. Five-run mean $\pm$ standard deviation.

Stage	Time (hours)	Share (%)
Frontier scoring	0.39 ± 0.31	6.39 ± 3.86
Prototype refresh	0.19 ± 0.15	3.23 ± 1.73
Embedding training	0.68 ± 0.48	11.72 ± 5.08
Total	1.26 ± 0.93	21.33 ± 10.66

Acceptance-check scaling. We group the 470 candidate checks from the five online Wonderland runs by the size of the accumulated explained set (Table 22). The check median is 2.99 seconds

overall and 85.00 seconds in the largest bucket; candidate program complexity also increases during learning. Full-set checking is therefore a non-trivial local CPU cost at large evidence-set sizes. It is performed when evaluating a proposed update, rather than at every interaction, and consumes no additional API tokens. The largest bucket contains 53 of the 470 checks. Meanwhile, median tokens per call rise from 8,786 to 36,354 across the smallest and largest buckets. The update prompt contains at most three selected broken examples, so it does not include the entire growing evidence set; program size and example length can still increase context cost.

Table 22: Acceptance-check scaling. Each entry is median (90th percentile); n counts candidate checks, not independent runs.

Explained-set size	n	Check time (s)	AST nodes	Tokens / call
$[0, 10)$	14	0.01 (0.02)	1,065.5 (1,295.9)	8,786 (14,952)
$[10, 100)$	35	0.04 (0.07)	1,632.0 (2,251.4)	17,880 (35,576)
$[100, 1{,}000)$	81	0.17 (0.23)	3,178.0 (4,043.0)	23,964 (58,948)
$[1{,}000, 10{,}000)$	89	0.63 (1.72)	4,371.0 (5,246.6)	20,676 (57,025)
$[10{,}000, 10^5)$	198	5.84 (16.41)	6,138.0 (8,544.2)	32,163 (59,127)
$[10^5, 10^6)$	53	85.00 (244.60)	7,507.0 (10,064.8)	36,354 (72,574)
All checks	470	2.99 (35.47)	5,126.0 (8,513.3)	25,833 (60,307)

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction describe a general method for online executable world-model learning and a controlled probe of lexical-prior dependence. Five-run Baba and ALFWorld results support the stated scope. The ablations report overall trends as class structure and exploration signals are added, supported by trend tests.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion limits the method to deterministic, fully observable symbolic settings with exact execution checks. Appendix A.10 discusses full-set checking and embedding overhead; qualitative examples show concrete prediction failures. ALFWorld does not establish applicability to raw visual or partially observed inputs.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results requiring proofs. The formal setup and information-theoretic view are used to define the problem and motivate the frontier score, not to state theorems.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper specifies the benchmark construction, train/test level splits, evaluation datasets, metrics, LLM-call budgets, termination criteria, prompt template, and implementation details needed to reproduce the main experiments. Additional dataset statistics and evaluation protocols are provided in Appendix A.7.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide the source code, released transition datasets, data-generation and evaluation scripts, and reproduction instructions in the supplementary material. The paper documents the dataset construction, level splits, metrics, LLM-call budgets, and evaluation protocol needed to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 and Appendix A.7 describe the Baba splits, metrics, budgets, embedding architecture, training schedule, and decoding settings. Appendix A.3 reports the coefficient sweep and default setting; Appendix A.8 gives the ALFWorld protocol and curriculum diagnostic.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: The main comparisons and ablations report means and standard deviations over five independent runs, with the run as the unit of analysis. Appendix A.3 specifies exact pairwise permutation and trend tests and their Holm corrections. Individual ablation comparisons are not significant after correction; the three ordered trends are. Additional analyses state their representative-run configuration.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A.7 specifies hardware and embedding training settings. Appendix A.10 reports token usage, relative weighted cost, embedding-stage elapsed times, and acceptance-check scaling; Appendix A.8 separately reports ALFWorld costs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses controlled symbolic game environments and archived transition data, and does not involve sensitive personal data, human-subject experimentation, or deployment on real users. The work is intended as foundational research on executable world-model learning.

   Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work is foundational research on online executable world-model learning in controlled symbolic game environments, with no deployment, human subjects, personal data, or application-specific decision making. We therefore do not identify direct societal impacts beyond the general indirect effects of progress in model-based AI systems.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The released assets consist of source code and controlled symbolic game transition datasets, not high-risk pretrained models, generative models, scraped web data, or sensitive personal data. We therefore do not identify misuse risks that require special release safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We credit *Baba Is You* as the source of the underlying game assets and level/map data used in the benchmark. We obtained permission from the game's developer to use these assets and map data for this research, and the submitted code/data package includes the corresponding attribution and usage information.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The submitted assets include the *Baba in Wonderland* benchmark code and transition datasets. The paper documents the remapping, state/action representation, level splits, dataset construction procedure, dataset statistics, and evaluation protocol, with additional reproduction details provided in the code package.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subject experiments. Human-play solution trajectories are used only as archived gameplay transition data and do not include participant studies, compensation, or personal information.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or human-subject research requiring IRB approval or equivalent review. No personal data or participant intervention is used.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of Alice's Programmer, and the paper describes their use for executable program updates, reports LLM-call budgets, includes the program-update prompt template, and evaluates backbone variation in the appendix.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.