CEDAR: Automata as Verifiable Interfaces for Language-Guided Embodied Action

Lekai Chen, Alvaro Velasquez and Ashutosh Trivedi
Department of Computer Science
University of Colorado Boulder, USA
{lekai.chen, alvaro.velasquez, ashutosh.trivedi}@colorado.edu

Abstract

Natural-language (NL) tasking of embodied agents is rarely just goal specification: users also impose constraints that must persist while the world changes. Code-generating LLM agents can produce plausible behaviors for such instructions, but their free-form programs provide no stable object to verify, compose with new constraints, or repair from a failing trace. We present CEDAR, a counterexample-guided framework that grounds instructions as regular languages over environment event traces. CEDAR uses a language model for semantic judgments and execution traces for correction, then represents both skills and specifications as deterministic finite automata. This turns constraints into executable finite-state objects: a learned skill can be intersected with a learned *sleep at night* or *stay in this biome* specification, yielding a controller that enforces the learned constraint by construction rather than by repeated prompting. In Minecraft, with the same simulator/API observations available to a program-generating baseline, CEDAR maintains temporal and spatial constraints that the baseline fails to preserve and amortizes reuse of learned skills, reducing cumulative LLM queries. These results suggest that regular languages offer a practical verification layer between natural-language instructions and embodied-agent policies.

1 Introduction

Natural language is an attractive interface for tasking embodied agents, but instructions rarely describe only a terminal goal. They also encode temporal, spatial, and safety constraints: *collect diamonds but sleep at night*; *explore the world but stay in the windswept forest*; *mine minerals only at night*. A useful agent must not merely execute a plausible program for the main goal; it must maintain a persistent interpretation of the user's constraints while acting in a changing environment.

Recent language-conditioned embodied systems address decision making through code generation, goal decomposition, skill storage, interleaved reasoning and action, and direct vision-language-action decoding (Yao et al., 2023; Wang et al., 2023; Zhu et al., 2023; Liu et al., 2023; Wang et al., 2024; Zhang et al., 2026). Beyond text-based agents, recent multimodal foundation models increasingly support long-horizon robotic control, spatial and visual reasoning, perceptual assessment, 3D understanding and generation, and fine-grained behavioral analysis (Liu et al., 2026b; Shen et al., 2026b; Fu et al., 2025; Li et al., 2026; Yu et al., 2025, 2026a; Shen et al., 2026a). These systems are flexible, but their learned skills are usually free-form programs or text memories. Such representations are hard to inspect formally. A generated program may satisfy a constraint once and then forget it; two skills may be hard to compose safely; and a failure trajectory may not provide a precise object to repair.

This paper studies a different interface between language and action: grounding a natural-language instruction into a *regular language over environment event traces*. Instead of treating the LLM as the final policy, we use it as a noisy semantic teacher. The LLM helps select a compact alphabet of action, object, and event-predicate symbols and answers membership queries about whether a trace is consistent with an instruction. A DFA learner builds an explicit automaton, while an environment-backed equivalence oracle tests the automaton and returns counterexamples. The resulting controller is less expressive than arbitrary code, but it is minimal up to isomorphism for the learned regular language, can be checked by replaying event traces, and is closed under operations such as intersection and concatenation.

We instantiate this idea in CEDAR, a Counterexample-Driven Agent with Regular-language controllers. CEDAR represents both

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 13694–13710
October 24-29, 2026 ©2026 Association for Computational Linguistics

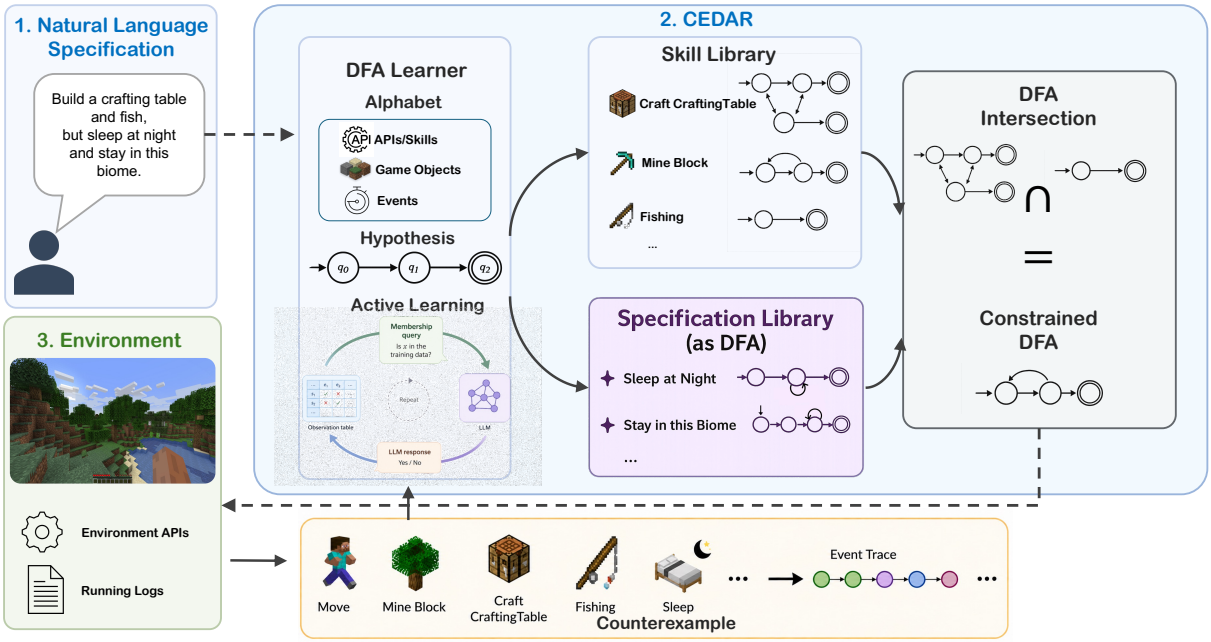


Figure 1: **CEDAR workflow.** The system grounds a natural-language goal or constraint into a compact symbolic alphabet, learns skill and specification DFAs using LLM membership queries (MQs) and environment or human counterexamples (CEs), stores reusable skill automata, and verifies execution by DFA composition. EQ denotes equivalence-query feedback from execution or human inspection.

skills and *specifications* as DFAs. Skill DFAs define executable progress toward goals, while specification DFAs encode constraints induced by natural language. A learned skill and specification can be combined through product intersection; prerequisite and follow-up skills can be combined through concatenation; and a failing execution trace can be returned to the learner as a counterexample. When a new task resembles a stored one, CEDAR reuses the DFA through template adaptation over a substituted verb–object alphabet.

A common point of confusion is the word *monitor*. In this paper, an event predicate or logger monitor is a read-only function over simulator/API state, such as time of day, biome, inventory, or whether an interaction changed the world. It is not a hidden perception module and not a hand-written safety shield. The code generated by VOYAGER has access to the same underlying state variables through the environment API. The difference is representational and architectural: CEDAR converts natural-language goals and constraints into persistent finite-state objects that can be intersected, checked, reused, and repaired, whereas free-form code skills do not expose an formal language-level operation analogous to DFA intersection.

To our knowledge, CEDAR is the first embodied-agent framework to use regular languages as a shared executable representation for both learned skills and learned natural-language constraints. This shared representation enables product intersection, concatenation, skill reuse under newly supplied constraints, and counterexample-guided relearning within one agent architecture. The main contribution is therefore not a new DFA-learning algorithm. CAPAL, active automata learning, and the use of an LLM as a semantic teacher build on prior work. The Minecraft experiments evaluate persistent constraint enforcement and amortized skill reuse. The iTHOR study evaluates only whether alphabet grounding and environment-backed counterexample repair can be instantiated in another simulator.

2 Task Formulation

We formulate instruction following as language grounding over event traces. Let $\mathcal{E}$ be an environment. At each time step, the agent executes low-level actions and the environment exposes structured observations through its simulator/API: inventory, time, biome, and nearby objects. CEDAR uses deterministic logger predicates over these same observations to emit symbolic events. A word extractor maps a trajectory of observations τ to a finite word $w = \phi(\tau) \in \Sigma^*$ over a task-specific alphabet Σ.

Alphabet. The global alphabet is a disjoint union

$$\Sigma_{\text{global}} = AP_{\text{act}} \uplus AP_{\text{obj}} \uplus AP_{\text{evt}}, \quad (1)$$

where AP_{act} contains control primitives such as `mine`, `craft`, and `place`; AP_{obj} contains objects such as `diamond_ore` or `bed`; and AP_{evt} contains event-predicate symbols such as `time=night`, `in_biome(windswept_forest)`, and `has_`k`(item)`. A symbol is therefore either an executable action-object event or an observed predicate emitted by the logger. The logger predicate must be backed by an implemented environment query; the LLM selects among available predicates but does not create new sensors.

Instructions as languages. A user instruction x induces two languages over traces: a goal language $L_g(x) \subseteq \Sigma^*$ and a constraint language $L_c(x) \subseteq \Sigma^*$. For example, in *collect diamonds but sleep at night*, the goal language accepts traces that reach diamond collection, while the constraint language rejects traces in which the agent continues working through night without sleeping. CEDAR learns DFA hypotheses A_g and A_c for these languages. A DFA is $A = (Q, \Sigma, \delta, q_0, F)$, with states Q, transition function $\delta : Q \times \Sigma \to Q$, initial state $q_0 \in Q$, and accepting states $F \subseteq Q$.

Queries and counterexamples. We use active DFA learning algorithm CAPAL in a setting that is robust to persistent noise (Angluin, 1987; Chen et al., 2026). A membership query (MQ) asks whether a word w should be accepted. In CEDAR, an LLM answers MQs from the natural-language instruction and retrieved symbol descriptions. An equivalence query (EQ) asks whether the current DFA matches the target. For skill DFAs, the environment approximates the EQ oracle: the agent executes the DFA, and receives a CE if accepted behavior does not achieve the intended outcome or rejected behavior is feasible. For high-level specifications, a human may also provide CEs through annotation, demonstration, or direct trace judgment.

3 CEDAR

The key design choice of CEDAR is to separate semantic grounding from control structure. The LLM supplies semantic judgments over language and symbols; the DFA supplies the persistent controller and formal operations. This separation is what makes a natural-language constraint such as *do not work at night* persist throughout execution rather than appearing only as a line of prompt context or a one-time generated subtask. Figure 1 summarizes CEDAR.

Term	Meaning in CEDAR
Alphabet Σ	Task-specific set of actions, objects, and event-predicate symbols used to write traces.
Word w	Sequence of symbols in Σ extracted from environment logs.
Skill DFA	Automaton whose accepted words correspond to successful execution of a task skill.
Specification DFA	Automaton whose accepted words satisfy a persistent natural-language constraint.
MQ	Query asking whether a word is consistent with an instruction; answered by the LLM.
EQ/CE	Test of a hypothesis DFA; a mismatch yields a counterexample trace.
Regular restriction	The finite-state behavioral language imposed by a specification DFA; this is a learned formal constraint, not an extra sensor.

Table 1: Language-grounding formulation.

3.1 Alphabet Construction

Learning over Σ_{global} is infeasible in an open-world environment with many APIs and objects, so CEDAR constructs a task-specific sub-alphabet Σ before DFA learning. The important assumption is not that CEDAR receives extra predefined observations, but that the environment already exposes structured state through the same simulator/API used by the code-generation baselines. Each candidate symbol in Σ_{global} has two parts: a natural-language description used for retrieval and an executable binding to an environment query, action primitive, or logger predicate. The LLM may select and prune symbols, but it cannot create a sensor from raw pixels or use a predicate that lacks an implementation.

Our Minecraft global inventory contains 1,429 implemented symbols: 16 action primitives, 9 environment-event predicates, and 1,404 object symbols. The alphabet constructor embeds the instruction, retrieves candidate verbs, objects, and event predicates from this inventory, and asks the LLM to prune them into the finite alphabet used by the active learner. Evaluation against target alphabets from 44 validated skill DFAs yields absolute accuracy 0.9372 ± 0.10 and overlap coefficient 0.9208 ± 0.10. The prompt and complete retrieval

analysis are reported in Appendix B.

If a required predicate is not implemented in Σ_{global}, CEDAR cannot express the intended regular language, just as a code-generating agent cannot reliably condition on an unavailable simulator variable. If the predicate exists globally but is omitted from the retrieved sub-alphabet, execution may expose the omission when the resulting behavior conflicts with an outcome recognized by the approximate equivalence oracle. In that case, CEDAR expands the alphabet and restarts learning. This mechanism is not complete: an omitted predicate may prevent the learned DFA from representing part of the instruction while the available oracle nevertheless accepts the resulting behavior. The DFA can then reach an accepting state despite failing to capture the full intended semantics.

3.2 Active DFA Learning under Noise

For a fixed instruction and alphabet, the learner repeatedly asks MQs, updates a DFA hypothesis, and tests it through execution. The LLM is not assumed to be a perfect oracle. We follow the probabilistic MAT view and use the CAPAL persistent-noise-aware active DFA-learning procedure (Chen et al., 2026): repeated and inconsistent MQ answers are reconciled in the query cache, while environment CEs correct systematic semantic errors. The system-level instantiation, including cache resolution, hypothesis construction, equivalence testing, is given in Appendix G.

For skill learning, the EQ oracle is implemented by execution. The current DFA is converted into a policy by choosing a shortest path from the current automaton state to an accepting state. If an invoked transition is absent from logs or fails to produce the expected event, the edge is temporarily disabled and the path is recomputed. If no accepting path remains, the trace is stored as a negative CE. If the DFA reaches an accepting state but the environment does not satisfy the intended goal, the accepted trace is likewise returned as a CE. The CE is passed to CAPAL's refinement routine, which updates the learner's observation storage and reconstructs the next hypothesis. Learning terminates when execution succeeds without producing a CE within the interaction budget.

3.3 Skill Storage, Retrieval, and Adaptation

A learned skill is stored as

$$s = \langle A, v, n, E, D \rangle, \tag{2}$$

where A is the DFA, $v \in AP_{\text{act}}$ is a verb, $n \in AP_{\text{obj}}$ is an object, $E \subseteq AP_{\text{evt}}$ is the set of success events, and $D \subseteq \Sigma^* \times \{0, 1\}$ is the labeled evidence used to learn the DFA. The evidence set is the union of MQ labels and EQ counterexamples.

For a new query (v', n'), the Skill Manager first attempts exact symbolic lookup. If $v' = v$ and $n' = n$, the stored DFA is reused directly. If the verb matches but the object differs, the manager treats the existing DFA as a template: it substitutes noun-specific symbols, updates examples accordingly, and refines the adapted DFA through the active learner. If the verb differs, the system learns a new

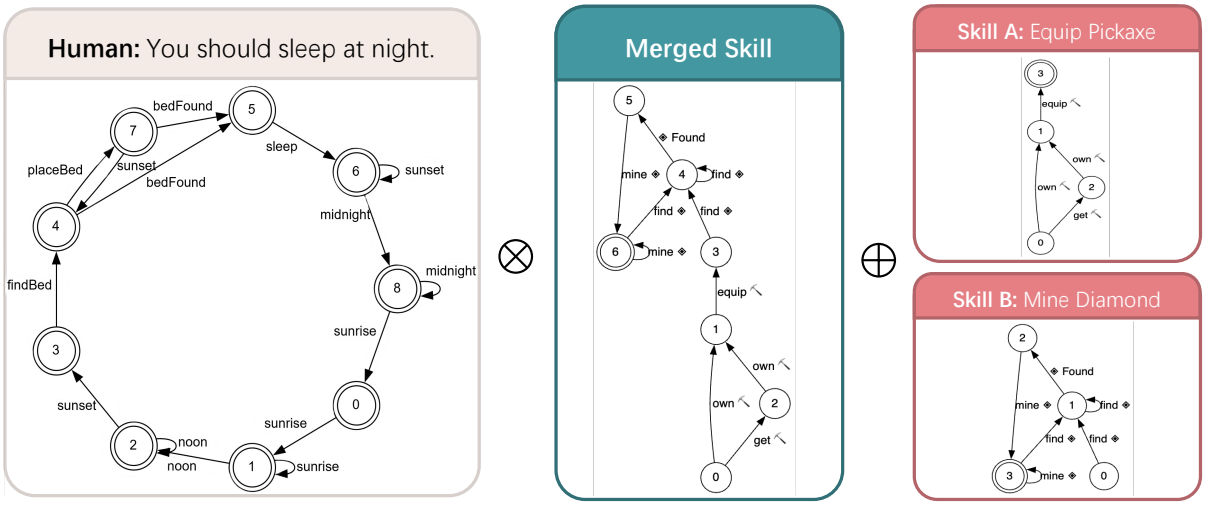


Figure 2: **DFA composition for constraint enforcement.** A learned specification DFA such as "sleep at night" is intersected with a learned skill DFA such as "mine diamond ore." The resulting controller accepts exactly the traces accepted by both learned automata. Rejecting sinks and nonessential transition detail are omitted for readability.

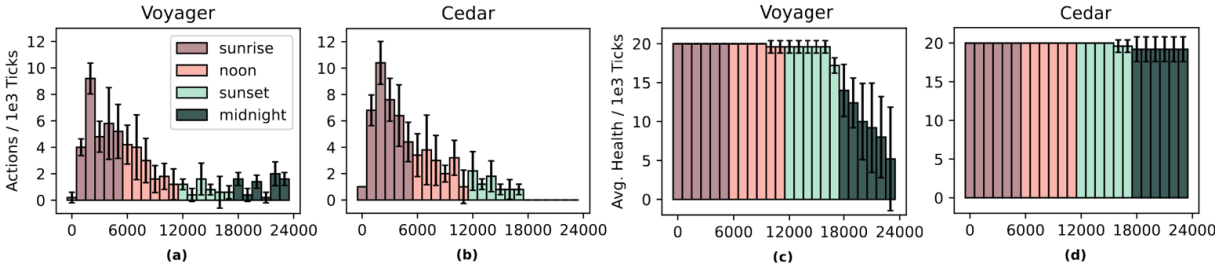


Figure 3: **Sleep-at-night constraint.** Action counts and average health for VOYAGER and CEDAR over repeated trials lasting three Minecraft days (aggregated over 24,000 ticks). Health denotes the averaged exact health value in the gameplay. Curves and error regions denote means and standard deviations. CEDAR suppresses night-time work because the constraint remains active throughout execution.

skill. This symbolic reuse is the main source of amortized efficiency.

3.4 Verifying Natural-Language Constraints

Specifications are learned as DFAs with the same procedure, except that human counterexamples may be used when the intended constraint is ambiguous. The specification DFA is the persistent constraint object: it is learned from natural language and counterexamples, not hand-coded as an external guard. Once a specification DFA A_c and a skill DFA A_g are available, CEDAR enforces their conjunction by product intersection:

$$L(A_g \cap A_c) = L(A_g) \cap L(A_c). \tag{3}$$

The composed automaton accepts exactly the traces accepted by both learned automata. This is an exact guarantee with respect to A_g and A_c; it is not independently a certificate that either learned automaton faithfully captures every clause of the original natural-language instruction. Appendix J reports the component-state counts, the Cartesian-product upper bounds, etc. A code-generating baseline can be prompted with the same natural-language constraint, and in our experiments it is; however, its stored skills are free-form programs rather than formal languages, so there is no exact operation analogous to $L(A_g) \cap L(A_c)$ for reusing a previously learned skill under a newly supplied constraint. Section 4 separately tests whether adding the same runtime assertion-and-repair condition to VOYAGER and CODEACT closes this gap.

CEDAR also chains skills through concatenation. If A_1 accepts traces for a prerequisite skill and A_2 accepts traces for a follow-up skill, accepting states of A_1 are connected to the initial state of A_2. Unlike unconstrained sequential calls to code skills, the DFA composition exposes whether the first skill actually reached an accepting condition before the second is invoked.

4 Experiments

We evaluate four questions. **Q1:** Under matched simulator/API observations, can regular-language controllers maintain natural-language constraints during execution? **Q2:** Does DFA skill reuse improve goal completion and LLM query cost? **Q3:** Are the LLM-grounded alphabet and counterexample loop reliable enough for practical learning? **Q4:** Can the same grounding pipeline be instantiated outside Minecraft?

The primary environment is Minecraft. We compare against VOYAGER (Wang et al., 2023) and CODEACT (Wang et al., 2024), including versions of each method with and without an assertion-and-repair mechanism. All methods receive the same task instruction and access to the same underlying simulator/API state variables. For the VOYAGER comparisons, we use the same map, spawn location, simulator APIs, and task instruction. CEDAR's event predicates are deterministic recodings of state variables available through those APIs and do not provide additional perceptual information.

Human CEs are not used for task-completion skill learning, the code-baseline comparisons, or the iTHOR experiments. They are used only in the high-level specification demonstrations, for which Appendix D reports the feedback modality, number of interventions, annotation source, and cost. We evaluate five iTHOR tasks as a feasibility study of alphabet grounding and environment-backed CE repair, not as a comparative benchmark or as evidence of cross-simulator skill reuse or specification intersection. The LLM components use GPT-5.5 for task decomposition and MQs, GPT-5.4-mini for JSON translation, and `text-embedding-3-large`

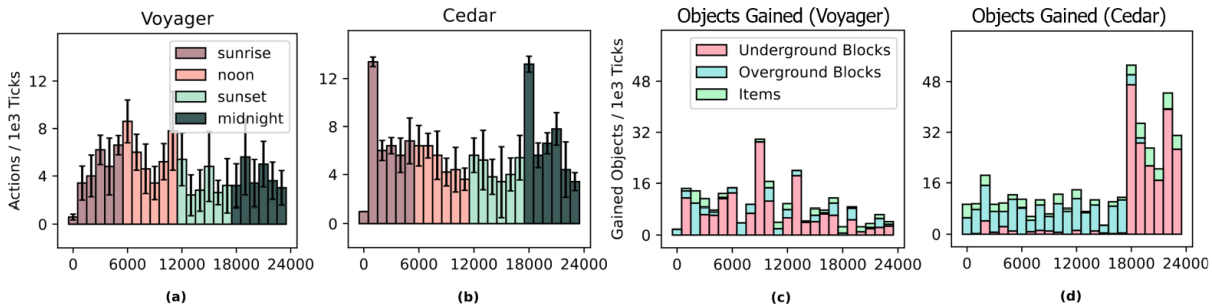


Figure 4: **Mine-minerals-only-at-night constraint.** Temporal distribution of actions and collected objects over five trials per method, with a fixed budget of 24,000 simulator ticks per trial. The direct compliance metric—the fraction of mineral-mining actions executed outside the night interval—is reported in Table 3.

Method	Found diamond	Constraint compliant	Joint success	Mean health ↑
VOYAGER without assertions	4/5	0/5	0/5	4.0 ± 3.16
VOYAGER with assertions	4/5	2/5	0/5	8.4 ± 4.56
CODEACT without assertions	3/5	1/5	0/5	4.8 ± 3.03
CODEACT with assertions	0/5	3/5	0/5	12.0 ± 5.83
CEDAR	**5/5**	**5/5**	**5/5**	$\mathbf{18.4 \pm 2.61}$

Table 2: **Sleep-at-night assertion comparison.** Exact 95% binomial confidence intervals are reported in Appendix H. Because $n = 5$, the comparison is interpreted descriptively rather than as a high-powered estimate of population-level success probabilities.

for embeddings.

4.1 Constraint Following in Minecraft

Sleep at night. The instruction is: *craft a diamond pickaxe and keep collecting diamonds; sleep at night; you are given a bed*. We evaluate VOYAGER and CODEACT both with and without the same assertion `night -> sleep before controller yield`, and compare them with CEDAR's learned specification-DFA composition. Consequently, a reused or long-running code skill may continue executing after night begins until control returns to the assertion boundary.Table 2 reports task completion, constraint compliance, joint success, and health over five trials per condition. A run is counted as successful if it finds a diamond, as constraint-compliant if it doesn't violate any assertions programmed by experts, and as jointly successful only if both events occur in the same trajectory. Health is computed as terminal health at corresponding trajectory ticks.

Runtime assertions improve local compliance, but none of the four code conditions achieves joint task-and-constraint success in the five evaluated trials. Trace inspection indicates that the assertions can repair a violation only when control returns to the assertion boundary: plans may treat sleep as a one-time subgoal, and reused or long-running skills may continue after night begins. CEDAR instead includes the learned specification state in the composed controller throughout execution.

Method	Actions	Underground	Overground	Items	Objects
VOYAGER	106 ± 4.6	157 ± 15	48.6 ± 8.7	24.8 ± 6	229 ± 14
CEDAR	137 ± 12	189 ± 30	141 ± 17	57.4 ± 4.7	387 ± 13

Table 3: Minecraft instruction following for "mine minerals only at night." Values are mean ± standard deviation for the corresponding recorded events.

Mine minerals only at night. The instruction is: *explore the world and collect as many different items as possible, but dig for minerals like iron and diamond only at night*. Both methods receive a fixed budget of 24,000 simulator ticks per trial. We measure constraint compliance through the daytime mineral-mining action counts. Table 3 summarizes object collection and Figure 4 shows the temporal distribution of actions and collected objects. CEDAR performs more total actions, collects more underground and overground objects under the reported protocol, and obtains more total items. The result reflects the difference between a one-time subtask and a persistent temporal restriction: CEDAR's higher action count reflects its use of daytime for exploration and nighttime for mineral extraction under the fixed horizon.

Stay in a requested biome. For the instruction *explore the world but stay in the windswept forest*,

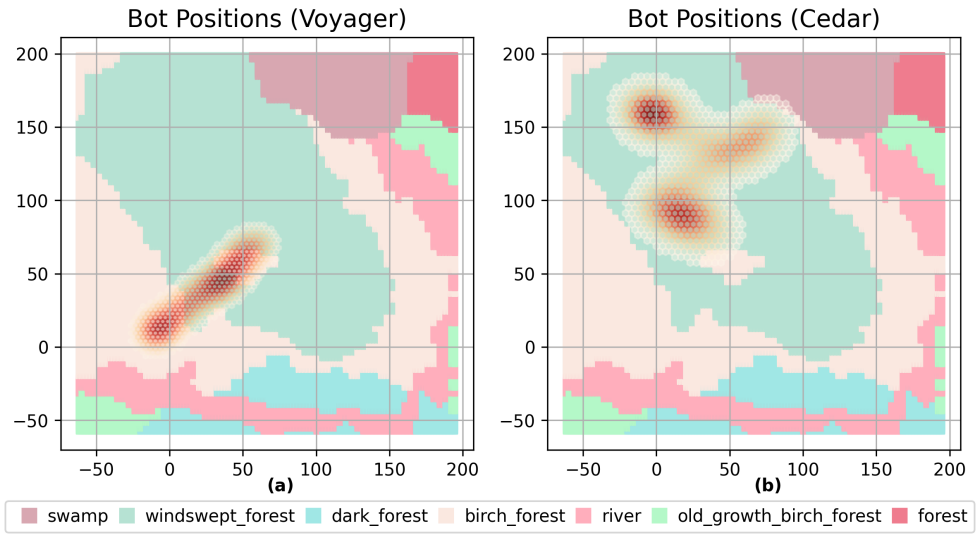


Figure 5: **Biome constraint.** Background colors denote Minecraft biomes and heatmap intensity denotes bot activity. CEDAR remains in the requested windswept forest region, while VOYAGER moves across biome boundaries. The heatmaps show an aggregation of location visit frequencies over 5 trials.

CEDAR includes biome-predicate symbols in the alphabet and learns a spatial constraint DFA. The biome predicate is obtained from the same simulator state available to both agents; CEDAR differs by turning it into a persistent formal constraint. The baseline traverses multiple biomes, while CEDAR restricts activity to the target biome. Figure 5 shows the resulting activity heatmaps. This experiment shows that the same language-grounding mechanism handles non-temporal event predicates.

4.2 Goal Completion and Skill Reuse

Table 4 reports prompting iterations and success rates for Minecraft crafting tasks. Without a skill library, CEDAR incurs additional interactions because it must learn and test a DFA. With a learned library, CEDAR adapts stored DFAs instead of repeatedly prompting for new code. Among successful trajectories, CEDAR with a skill library uses fewer mean prompting iterations than VOYAGER with a skill library on four of the five tasks. These means are conditioned on success, however, and therefore summarize different trajectory subsets when success rates differ. They are not unconditional efficiency estimates. Appendix I reports exact confidence intervals for the success rates, while Appendix A provides the cleaner cumulative-query accounting for the amortization claim.

Appendix A reports cumulative LLM-query and wall-clock accounting for the amortization claim; these resource-budget measurements are kept separate from statistical task-performance estimates.

4.3 Grounding Reliability

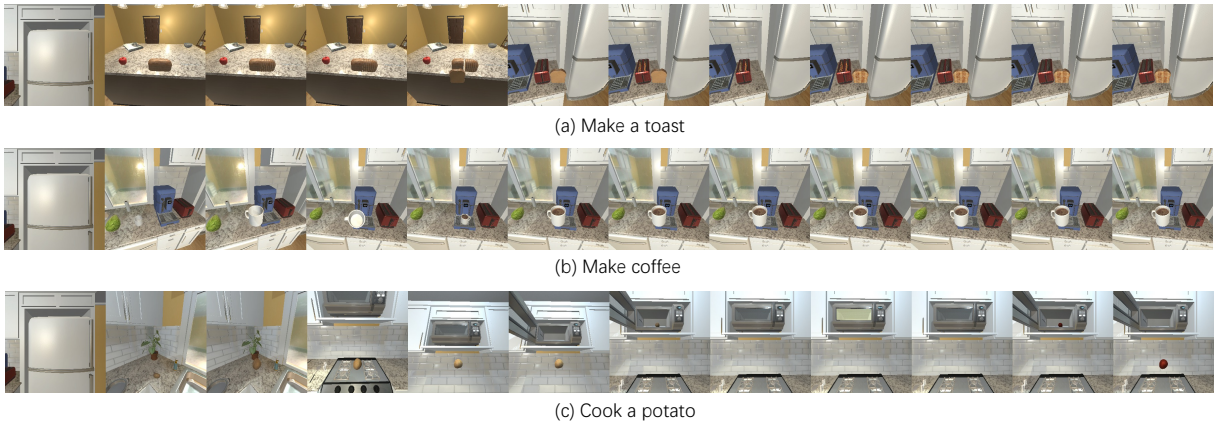
The alphabet and CCE loop are explicit dependencies of the method, so we analyze them separately from end-to-end task success. Against their validated target alphabets, retrieval achieves absolute accuracy 0.9372 ± 0.10 and overlap coefficient 0.9208 ± 0.10. Appendix B reports the complete retrieval analysis. Appendix C.2 separately evaluates environment-backed CE discovery by perturbing correct skill DFAs. The average reported discovery accuracy is 0.9358. Together, these analyses support the practical use of an LLM semantic teacher corrected through environment feedback while leaving absent, rare, or noisy predicates as central failure modes. In particular, neither analysis establishes that every instruction-relevant predicate will be retrieved or that a silent alphabet omission will always be detected.

4.4 iTHOR Feasibility Study

To test whether task-specific alphabet grounding and environment-backed counterexample repair can be instantiated outside Minecraft, we run five iTHOR tasks: MAKE TOAST, MAKE COFFEE, COOK POTATO, MAKE SALAD, and STORE PLATE IN FRIDGE. They do not evaluate persistent constraint enforcement, skill reuse, or amortized reuse outside Minecraft. Each task uses task-specific alphabet construction, executable predicate bindings, and an execution-backed counterexample loop. Each skill requires two LLM calls: one for

Method	Wooden Pickaxe	Iron Pickaxe	Diamond Pickaxe	Lava Bucket	Compass
VOYAGER w/o S.L.	6.9 ± 2.1 (25/25)	30.4 ± 4.5 (25/25)	33.4 ± 10.4 (11/25)	26.4 ± 7.2 (19/25)	25.9 ± 2.9 (17/25)
VOYAGER	4.2 ± 3.5 (25/25)	16.2 ± 3.8 (25/25)	28.9 ± 10.7 (17/25)	22.3 ± 5 (25/25)	17.7 ± 2.8 (25/25)
CEDAR w/o S.L.	6.6 ± 2.8 (25/25)	30.4 ± 3.3 (25/25)	45.9 ± 11.1 (18/25)	27.2 ± 5.6 (25/25)	29.8 ± 2.1 (9/25)
CEDAR	5.5 ± 4 (25/25)	9.6 ± 6 (25/25)	19.8 ± 6.5 (23/25)	10.5 ± 8.5 (25/25)	10 ± 2.7 (25/25)

Table 4: **Minecraft goal completion.** Each method–task condition contains 25 trials. Entries report prompting iterations as mean $\pm$ standard deviation over successful trajectories only. Exact 95% binomial confidence intervals for success rates are reported in Appendix I. S.L. denotes a skill library.



(a) Make a toast

(b) Make coffee

(c) Cook a potato

Figure 6: **iTHOR grounding-and-repair feasibility rollouts.**Time-lapse frames for making toast, making coffee, and cooking a potato. The potato task initially produced a CE due to camera perspective; adding a `LOOKUP` action repaired the DFA without extra LLM queries. These rollouts show that the pipeline can run outside Minecraft, but they are not used as a comparative benchmark. Results are based on 3 runs.

sub-alphabet construction and one to synthesize the code-based oracle program that binds logger predicates and action stubs. Four tasks succeed in a single pass. COOK POTATO initially fails because of camera perspective; the approximate equivalence oracle returns a counterexample, and inserting a `LOOKUP` action repairs the DFA without additional LLM calls. Representative rollouts are shown in Figure 6. We do not present VOYAGER results in iTHOR because its released implementation is coupled to Minecraft and Mineflayer APIs. Accordingly, all comparative claims, persistent constraint-enforcement claims, and skill-reuse claims in this paper are supported by the Minecraft evaluation.

5 When Do Regular Languages Suffice?

A central limitation of CEDAR is also its source of interpretability: DFAs recognize regular languages. They cannot count indefinitely, store unbounded memory, or implement arbitrary recursion as JavaScript programs can. We therefore characterize the intended use case.

Let Σ be a task alphabet extracted from implemented logger predicates, including thresholded predicates such as `has_3_cobblestone`, time predicates such as `time=night`, and spatial predicates such as `in_biome(x)`. If all numerical conditions needed for a skill are exposed as finite milestone predicates and each subtask completes or fails within a bounded event horizon H, then the successful traces of that subtask are regular. Intuitively, these predicates convert environment state into finite symbols, and the bounded horizon yields a finite control graph. Under these assumptions, intersection for constraint enforcement and concatenation for skill chaining preserve regularity.

This analysis clarifies both the power and the boundary of the approach. CEDAR is appropriate when instructions can be expressed through finite milestones and persistent temporal or spatial restrictions backed by available environment predicates. It is less appropriate when the task requires unbounded counting, arbitrary data structures, or long-range memory not captured by the logger. In such settings, DFAs may still be useful as learned constraints, but the skill representation should be extended to richer formal models such as counter automata, visibly pushdown automata, or recursive state machines.

6 Related Work

A related line translates natural-language instructions into temporal logic (Fuggitti and Chakraborti, 2023; Pan et al., 2023; Liu et al., 2024). These methods typically assume a proposition vocabulary and study formula translation using paired natural-language and formal specifications. CEDAR instead retrieves an API-backed task alphabet and learns an executable DFA through active judgments and execution CE. The supervision assumptions are therefore different, and we do not empirically establish superiority over NL-to-LTL translation systems. An NL-to-LTL front end could initialize a specification DFA when its output is regular or compilable to a finite monitor; subsequent CE-guided repair would remain complementary.

Automata have also verified LLM-generated robot scripts. Yang et al. (2024) translate them into automata and check them against external safety specifications. In contrast, CEDAR learns skill and NL constraint automata from semantic judgments and execution traces, stores them as executable agent representations, and composes them.

Classical active DFA learning identifies regular languages through membership and equivalence queries (Angluin, 1987). Recent work studies LLMs as probabilistic semantic teachers and persistent-noise-tolerant active DFA learning (Chen et al., 2026); related work also learns automata from NL oracles (Vazquez-Chanlatte et al., 2024). CEDAR inherits these learning ideas rather than introducing a new active automata-learning algorithm. Its contribution is their use within an embodied-agent architecture in which both reusable skills and learned language constraints share the same executable regular-language representation.

Xu et al. (2020) jointly infer reward machines and reinforcement- learning policies by collecting episodes whose observed rewards are inconsistent with a hypothesis reward machine and then relearning the hypothesis. CEDAR likewise uses execution inconsistencies as counterexamples, but addresses a different setting and interface: it constructs task-specific alphabets from simulator APIs, uses an LLM to judge NL trace membership, executes learned DFAs as controllers, and composes learned skill and specification automata. Counterexample-guided automaton refinement itself is therefore inherited rather than claimed as novel. Safe reinforcement learning and shielding constrain policies during learning or execution (Moos et al., 2022; Gu et al., 2024). These methods generally assume that the relevant constraint or shield is already available. CEDAR addresses the preceding grounding problem by learning a finite-state specification from natural language and counterexamples. A learned CEDAR specification DFA could in principle be used as a shield around another controller, but this paper does not compare against systems supplied with independently validated formal specifications and does not claim that its learned DFAs are certified safety specifications. Related work also highlights limitations of accuracy-only evaluation and introduces structural guarantees, trace-level diagnostics, and regime-aware evaluation for foundation models (Wang et al., 2026b; Sun et al., 2026; Wang et al., 2026a). More broadly, recent work improves LLM inference through self-improvement, learnability-aware trajectory selection, evidence routing, structured search, model routing, and cache reuse (Yu et al., 2026b,c; Zhao et al., 2026; Ye et al., 2026a; Guo et al., 2026; Ye et al., 2026b; Wen et al., 2026; Wei et al., 2026; Wei et al.); CEDAR is complementary in reducing the symbolic decision space through task-specific alphabet retrieval.

7 Conclusion

We presented CEDAR, the first framework that uses regular languages as a verifiable representation for learned embodied skills and natural-language constraints. Within the evaluated symbolic, event-driven setting, this shared representation makes learned instruction semantics explicit: skill and specification automata can be combined by product intersection, skills can be chained and reused, and failing execution traces can be returned as counterexamples for relearning. The Minecraft experiments provide evidence for persistent temporal and spatial constraint following, including relative to VOYAGER and CODEACT variants augmented with the same runtime assertion-and-repair condition. They also provide the evidence for amortized skill reuse. The iTHOR study demonstrates that the same pipeline can be instantiated in another simulator. Overall, the results support regular languages as a practical common substrate for learned skills and constraints when instructions can be represented through finite predicates.

Limitations

Evidence for persistent constraint enforcement, skill composition, and amortized reuse comes from Minecraft. The iTHOR study evaluates only alphabet grounding, executable predicate binding, and environment-backed counterexample repair. CAPAL assumes structured simulator/API observations and does not reliably extract predicates from raw visual input. The evaluated environments are discrete and symbol-rich; we claim no advantage over code or continuous-control policies requiring unbounded memory, arbitrary data structures, or fine-grained continuous state.

DFA intersection exactly enforces the languages represented by the learned automata, but fidelity to the user's instruction depends on the selected alphabet and the correctness of membership- and equivalence-query feedback. Missing predicates can silently omit intended clauses while still permitting acceptance. Execution-backed alphabet expansion may reveal some omissions, but it is incomplete, and we do not systematically measure their rate.

Incorrect human counterexamples can define the wrong learning target and steer CAPAL toward an incorrect automaton. Appendix D documents the human-feedback demonstrations, but this study does not establish robustness to incorrect or conflicting feedback. Where only one provider was used, inter-annotator sensitivity cannot be estimated.

Product automata have worst-case multiplicative state growth. Although the tested compositions remain small (Appendix J), this does not guarantee general scalability. Retrieval is evaluated over the current 1,429-symbol inventory without controlled variation of inventory size or target-automaton complexity.

Finally, learned specification DFAs are not certified safety specifications. They are insufficient for physical robots or other safety-critical systems without independently validated sensing, specifications, runtime enforcement, and fail-safe control. Exact composition cannot correct incorrectly grounded semantics.

Acknowledgments

This work was supported by DARPA award number HR0011261E001 and the NSF under CAREER Award CCF-2146563. Ashutosh Trivedi is a Royal Society Wolfson Visiting Fellow and gratefully acknowledges support from the Wolfson Foundation and the Royal Society.

References

Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106.

Chengkun Cai, Xu Zhao, Haoliang Liu, Zhongyu Jiang, Tianfang Zhang, Zongkai Wu, Jenq-Neng Hwang, and Lei Li. 2025. The Role of Deductive and Inductive Reasoning in Large Language Models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Lekai Chen, Ashutosh Trivedi, and Alvaro Velasquez. 2026. Towards persistent noise-tolerant active learning of regular languages with class query. In *The Fourteenth International Conference on Learning Representations*.

Xingyu Fu, Siyi Liu, Yinuo Xu, Pan Lu, Guangqiuse Hu, Tianbo Yang, Taran Anantasagar, Christopher Shen, Yikai Mao, Yuanzhe Liu, and 1 others. 2025. Learning human-perceived fakeness in ai-generated videos via multimodal llms. *arXiv preprint arXiv:2509.22646*.

Francesco Fuggitti and Tathagata Chakraborti. 2023. NL2LTL: A python package for converting natural language instructions to linear temporal logic formulas. In *Proceedings of the AAAI Conference on Artificial Intelligence, System Demonstrations*.

Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. 2024. A review of safe reinforcement learning: Methods, theory and applications. *arXiv preprint arXiv:2205.10330*.

Yu Guo, Shenghao Ye, Shuangwu Chen, Zijian Wen, Tao Zhang, Bai Qirui, Dong Jin, Yunpeng Hou, Huasen He, Jianyang, and Xiaobin Tan. 2026. Rethinking table pruning in TableQA: From sequential revisions to gold trajectory-supervised parallel search. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12960–12976, San Diego, California, United States. Association for Computational Linguistics.

Zhiyi Kuang, Ryan Rong, YuCheng Yuan, and Allen Nie. Learning game-playing agents with generative code optimization. In *ICML 2025 Workshop on Programmatic Representations for Agent Learning*.

Boyi Li, Yifan Shen, Yuanzhe Liu, Yifan Xu, Jiateng Liu, Xinzhuo Li, Zhengyuan Li, Jingyuan Zhu, Yunhan Zhong, Fangzhou Lan, and 1 others. 2026. Toward cognitive supersensing in multimodal large language model. *arXiv preprint arXiv:2602.01541*.

Haoliang Liu, Chengkun Cai, Xu Zhao, Han Zhu, Shizhou Huang, Xinglin Zhang, Tao Chen, Jenq-Neng Hwang, Zhang Huaping, and Lei Li. 2026a.

Bcl: Bayesian in-context learning framework for information extraction. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 31172–31189.

Jason Xinyu Liu, Ankit Shah, George Konidaris, Stefanie Tellex, and David Paulius. 2024. Lang2ltl-2: Grounding spatiotemporal navigation commands using large language and vision-language models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, and 1 others. 2026b. Palm: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. *arXiv preprint arXiv:2601.07060*.

Janosch Moos, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, and Jan Peters. 2022. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315.

Allen Nie, Xavier Daull, Zhiyi Kuang, Abhinav Akkiraju, Anish Chaudhuri, Max Piasevoli, Ryan Rong, YuCheng Yuan, Prerit Choudhary, Shannon Xiao, and 1 others. 2026. Understanding the challenges in iterative generative optimization with llms. *arXiv preprint arXiv:2603.23994*.

Jiayi Pan, Glen Chou, and Dmitry Berenson. 2023. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. In *IEEE International Conference on Robotics and Automation*.

Yifan Shen, Boyi Li, Meihuan Huang, Yuanzhe Liu, Xu Cao, Jinyang Jin, Zhengyuan Li, Anglin Liu, Junho Kim, Jingyuan Zhu, and 1 others. 2026a. Decoding children's gait behavior. *arXiv preprint arXiv:2608.00371*.

Yifan Shen, Yuanzhe Liu, Jingyuan Zhu, Xu Cao, Xiaofeng Zhang, Yixiao He, Wenming Ye, James Rehg, and Ismini Lourentzou. 2026b. Fine-grained preference optimization improves spatial reasoning in vlms. *Advances in Neural Information Processing Systems*, 38:17929–17960.

Jingzhe Shi, Qinwei Ma, Hongyi Liu, Hang Zhao, Jenq-Neng Hwang, and Lei Li. 2026. Intrinsic entropy of context length scaling in llms. In *The Fourteenth International Conference on Learning Representations*.

Xian Sun, Wei Gao, Yingshuo Wang, Lingdong Kong, Yanhang Li, Zhichao Fan, Zexin Zhuang, Wenlong Dong, Zhiyuan Zheng, Hrishikesh Paranjape, Abhishek Mandal, and Johnny R. Zhang. 2026. Beyond accuracy: Measuring bias acknowledgment in chain-of-thought reasoning for responsible ai evaluation. *Preprint*, arXiv:2606.15127. ICML Workshop on AI4GOOD.

Marcell Vazquez-Chanlatte, Karim Elmaaroufi, Stefan J. Witwicki, and Sanjit A. Seshia. 2024. l^*lm: Learning automata from examples using natural language oracles. *arXiv preprint arXiv:2402.07051*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. In *Advances in Neural Information Processing Systems*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Yingshuo Wang, Xian Sun, Lingdong Kong, Wei Gao, Yanhang Li, Zhichao Fan, and Zexin Zhuang. 2026a. Do time series foundation model benchmarks hide regime-dependent failures? evidence from traffic speed forecasting. *Preprint*, arXiv:2606.18367. ICML Workshop on Forecast.

Yingshuo Wang, Xian Sun, Yanhang Li, Zhichao Fan, and Zexin Zhuang. 2026b. Embedding foundation model predictions in discrete-choice models with structural guarantees. *Preprint*, arXiv:2606.26432.

Jiaqi Wei, Xuehang Guo, Pengfei Yu, Xiang Zhang, Wanli Ouyang, Siqi Sun, Qingyun Wang, and Chenyu You. 2026. When to think, when to speak: Learning disclosure policies for llm reasoning. *arXiv preprint arXiv:2605.03314*.

Jiaqi Wei, Hao Zhou, Xiang Zhang, Di Zhang, Zijie Qiu, Noah Wei, Jinzhe Li, Wanli Ouyang, and Siqi Sun. Retrieval is not enough: Enhancing rag through test-time critique and optimization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Zijian Wen, Tao Zhang, Shuangwu Chen, Shenghao Ye, Yu Guo, Qirui Chen, Jingxian Shuai, Yunpeng Hou, Huasen He, and Jianyang. 2026. SpecCache: Speculative KV cache reuse for efficient RAG serving. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18861–18871, San Diego, California, United States. Association for Computational Linguistics.

Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. 2020. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 590–598.

Yunhao Yang, Neel P. Bhatt, William Ward, Zichao Hu, Joydeep Biswas, and Ufuk Topcu. 2024. Joint verification and refinement of language models for safety-constrained planning. *arXiv preprint arXiv:2410.14865*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Shenghao Ye, Yu Guo, Dong Jin, Yuxiang Wang, Yikai Shen, Yunpeng Hou, Shuangwu Chen, Jianyang, and Xiaofeng Jiang. 2026a. When TableQA meets noise: A dual denoising framework for complex questions and large-scale tables. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24022–24045, San Diego, California, United States. Association for Computational Linguistics.

Shenghao Ye, Yuxiang Wang, Yu Guo, Dong Jin, Shuangwu Chen, and Jian Yang. 2026b. Rethinking stepwise model routing: A cost-efficient table reasoning perspective. *Preprint*, arXiv:2605.29319.

Tianjiao Yu, Xinzhuo Li, Yifan Shen, Yuanzhe Liu, and Ismini Lourentzou. 2025. Core3d: Collaborative reasoning as a foundation for 3d intelligence. *arXiv preprint arXiv:2512.12768*.

Tianjiao Yu, Xinzhuo Li, Yifan Shen, Onkar Susladkar, Yuanzhe Liu, Xiaona Zhou, and Ismini Lourentzou. 2026a. Elsa3d: Elastic semantic anchoring for unified 3d understanding and generation. *arXiv preprint arXiv:2607.06565*.

Tianrun Yu, Yuxiao Yang, Zhaoyang Wang, Kaixiang Zhao, Porter Jenkins, Xuchao Zhang, Chetan Bansal, Huaxiu Yao, and Weitong Zhang. 2026b. Provable and practical in-context policy optimization for self-improvement. *arXiv preprint arXiv:2603.01335*.

Tianrun Yu, Kaixiang Zhao, Chih-Chun Chen, Amanda Hughes, Taylor W Killian, Fenglong Ma, Weitong Zhang, and Porter Jenkins. 2026c. Lark: Learnability-grounded trajectory selection for efficient reasoning distillation. *arXiv preprint arXiv:2605.30651*.

Yucheng Yuan, Yuanfeng Ji, Zhongxiao Li, and Ruijiang Li. 2026. Sp-mind: An autonomous reasoning agent for spatial proteomics analysis. *arXiv preprint arXiv:2606.24235*.

Zhewei Zhang, Puyue Wang, Guanren Qiao, Yijie Weng, Jiawei Hu, Guo Li, Lujia Wang, Junyan Wang, Tao Gu, Hongliang Lu, Guiliang Liu, Hong Jia, and Xinhu Zheng. 2026. Lira: Local cross-layer information routing for vision-language-action decoding. *Preprint*, arXiv:2608.07596.

Kaixiang Zhao, Tianrun Yu, Shawn Huang, Porter Jenkins, Yushun Dong, and Amanda Hughes. 2026. Tiger: Traceable inference with graph-based evidence routing for mitigating hallucinations in multimodal generation. *arXiv preprint arXiv:2606.00232*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

A LLM Query and Runtime Cost

The cost of DFA learning is front-loaded. Once a DFA is learned, retrieval, intersection, and concatenation are symbolic operations. On the *Find a Diamond* milestone sequence, CEDAR uses more queries at the first logging milestone but fewer cumulative queries from the Cobblestone milestone onward, reaching the final diamond milestone with 37 LLM queries compared with 75 for VOYAGER (Table 5 and Figure 7). Wall-clock measurements show the same amortization effect: CEDAR incurs DFA construction time but has much lower skill-addition and total execution time under the reported protocol (Table 6). These resource-budget results are separate from statistical task-performance estimates: success or object-count claims should be supported by confidence intervals, while amortization claims are based on cumulative LLM-query and runtime accounting.

Method	Logs	Table	Cobble	Furnace	Iron	Diamond
VOYAGER	4	13	25	34	41	75
CEDAR	18	20	23	28	30	37

Table 5: Cumulative LLM queries on the *Find a Diamond* milestone sequence. Lower is better.

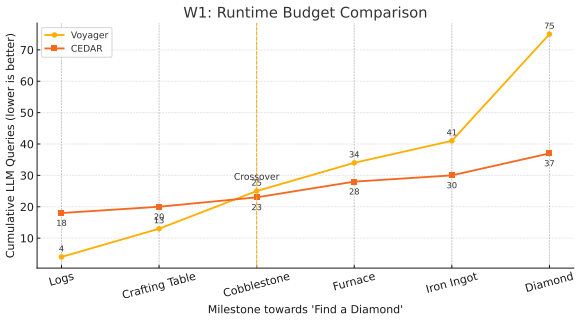


Figure 7: **Cumulative LLM-query budget** for the *Find a Diamond* milestone sequence. CEDAR pays an early alphabet-construction cost but amortizes it through DFA reuse and reaches the final milestone with fewer total queries.

Stage	VOYAGER	CEDAR
Task decomposition	2.749 ± 1.238	3.943 ± 2.134
Code/sample generation	6.381 ± 1.990	5.548 ± 1.876
Program description	2.384 ± 1.208	N/A
Skill addition	2.653 ± 1.228	0.021 ± 0.008
DFA construction	N/A	18.548 ± 14.289
Skill retrieval	0.323 ± 0.235	0.089 ± 0.586
Total execution	62.427 ± 55.834	33.101 ± 24.391

Table 6: Stage-wise wall-clock time in seconds, reported as mean $\pm$ standard deviation over 5 independent measurements per stage.

Task	LLM calls	Outcome
MAKE TOAST	2	success
MAKE COFFEE	2	success
COOK POTATO	2	success after CE fix
MAKE SALAD	2	success
STORE PLATE IN FRIDGE	2	success

Table 8: iTHOR query budget for the feasibility study. The counterexample fix for COOK POTATO did not require additional LLM calls. Each task is evaluated over 3 executions.

B Alphabet Retrieval Details

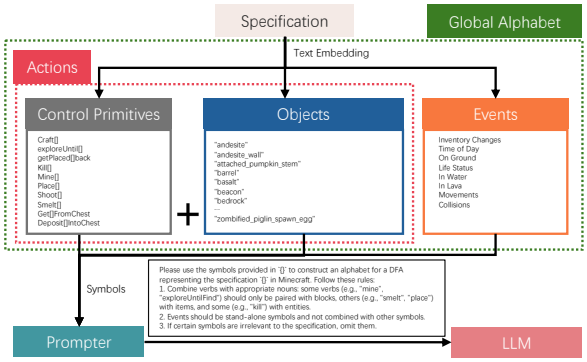


Figure 8: **RAG-based alphabet construction.** From the global action, object, and event-predicate alphabet, retrieval selects task-relevant symbols that the LLM prunes into the finite alphabet used for DFA learning. The selected predicates are symbolic recodings of existing simulator/API observations, not additional perceptual inputs.

The alphabet constructor retrieves candidate symbols with implemented environment bindings and asks the LLM to build a compact alphabet. We evaluate the retriever against target alphabets derived from 44 validated skill DFAs. Table 7 reports high absolute accuracy and overlap, indicating that the LLM usually reasons over a relevant subalphabet rather than the full global space. This is an analysis of symbol retrieval, not a claim that CEDAR invents new sensors.

Metric	Absolute Acc.	Overlap Coef.	Cosine Sim.
RAG	0.9372 ± 0.10	0.9208 ± 0.10	0.4500 ± 0.14

Table 7: RAG alphabet-construction performance over 44 validated skill DFAs.

C Additional Results

C.1 iTHOR Query Budget

C.2 Counterexample Discovery by Item

Item	Accuracy	Standard deviation
Dirt	0.9727	0.1629
Birch Log	0.8636	0.3432
Grass Block	1.0000	0.0000
Birch Leaves	0.9909	0.0949
Stone	0.9727	0.1629
Coal Ore	1.0000	0.0000
Iron Ore	1.0000	0.0000
Copper Ore	0.9909	0.0949
Gold Ore	0.9636	0.1872
Redstone Ore	0.9636	0.1872
Emerald Ore	0.4909	0.4999
Diamond Ore	0.9818	0.1336
Lapis Ore	0.9636	0.1872
Andesite	0.9818	0.1336
Granite	0.9636	0.1872
Sand	0.8727	0.3333
Average	0.9358	0.1692

Table 9: Counterexample discovery in Minecraft simulations after perturbing correct skill DFAs.

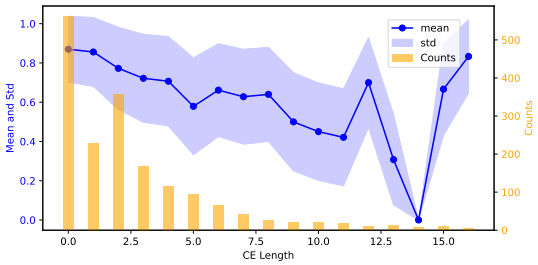


Figure 9: **Counterexample collection by CE length.** Shorter counterexamples are easier to surface during simulation; most observed CEs have length below seven, where collection probability remains high.

C.3 Expert DFA Reference

As an upper-bound diagnostic for the representation, manually designed DFA skills were tested on the same Minecraft tasks. This is not a learning baseline: it assumes expert construction and bypasses the language-grounding problem. The expert DFAs achieved 10/10 success for Wooden Pickaxe, Iron Pickaxe, Lava Bucket, and Compass, and 9/10 for Diamond Pickaxe, with the failure caused by map initialization.

D Human Counterexample Accounting

Human feedback is used only for the high-level specification demonstrations. It is not used for Minecraft task-completion skill learning, the code-baseline comparisons, or the iTHOR experiments. Table 10 reports the interventions actually used in the reported specification-learning runs.

Specification	Human CEs	Final DFA states
Sleep at night	8	9
Mining only at night	5	5
Stay in windswept forest	4	2

Table 10: Human counterexamples used for specification-DFA learning. The modality identifies trajectory annotation, demonstration logs, or direct formal trace judgment. Time reports the total annotation time used in the reported run rather than a theoretical intervention bound.

The reported specification feedback was supplied by on human expert (author). Consequently, the experiments do not measure inter-annotator disagreement or sensitivity to alternative interpretations. A semantically incorrect counterexample can define an incorrect target language and cause CAPAL to learn the wrong specification DFA.

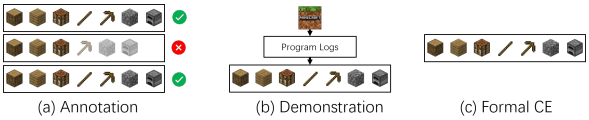


Figure 10: **Human counterexample interfaces.** Counterexamples can be supplied through trajectory annotation, demonstration logs, or direct formal words over the DFA alphabet.

E Prompt Sketch for Alphabet Construction

The RAG-style alphabet constructor retrieves candidate symbols with implemented environment bindings and asks the LLM to build a compact alphabet. A simplified prompt is shown below:

> Given the specification and the candidate symbols in braces, construct an alphabet for a DFA in Minecraft. Combine verbs with appropriate nouns: `mine` should be paired with blocks, `smelt` and `place` with items, and `kill` with entities. Event predicates should remain standalone symbols. Omit irrelevant symbols. Return only the selected symbols.

F Failure Modes

Missing predicates. If no implemented symbol represents a required milestone, the DFA cannot encode it. For example, a task requiring at least three cobblestones needs a finite event predicate

such as `has_3_cobblestone`. Without that predicate, the DFA may over-accept or fail to distinguish relevant progress states. If the predicate exists in the global inventory but is omitted from the retrieved task alphabet, an execution counterexample may expose the omission and trigger alphabet expansion. This recovery is not guaranteed: if the available oracle does not recognize the semantic mismatch, the incomplete DFA may still reach an accepting state. This is a logger/alphabet limitation rather than an observation-advantage assumption, since code-generating agents also require access to the relevant simulator variable.

Rare long-horizon counterexamples. Counterexamples become harder to discover when an incorrect transition appears only after a long execution prefix. Most simulated counterexamples are short, but rare long-horizon failures remain possible.

Shared semantic misunderstanding. The LLM and human feedback provider may share an incorrect interpretation of an ambiguous instruction. A semantically incorrect counterexample can then define the wrong target language and direct the active learner toward an incorrect automaton. Counterexamples help only when at least one oracle can recognize the mismatch. CEDAR does not claim robustness to systematically incorrect equivalence feedback.

More broadly, recent work studies LLM inference through Bayesian in-context adaptation, context-length scaling, and deductive–inductive reasoning (Liu et al., 2026a; Shi et al., 2026; Cai et al., 2025), while LLM-based agents increasingly employ code generation and optimization, goal decomposition, skill reuse, interleaved reasoning and action, and domain-specific tools (Yao et al., 2023; Wang et al., 2023; Zhu et al., 2023; Liu et al., 2023; Wang et al., 2024; Kuang et al.; Nie et al., 2026; Yuan et al., 2026). These directions are complementary to CEDAR, whose primary failure modes arise from symbolic grounding, counterexample discovery, and oracle correctness.

G CEDAR Instantiation of CAPAL

Algorithm 1 gives the system-level interaction between CEDAR and CAPAL. CAPAL maintains its active-learning state, including its prefix set, suffix sets, state partition, discrimination tree, and hypothesis DFA. CEDAR supplies semantic membership judgments, persistent query-cache evidence, and execution-backed counterexamples.

Algorithm 1 CEDAR instantiation of CAPAL

Require: Instruction x; alphabet Σ; learner $\mathcal{L}$; MQ budget B_{MQ}; execution/EQ budget B_{EQ}
Ensure: DFA A or budget-exhaustion report

Persistent MQ cache $C \leftarrow \varnothing$
Gold counterexample cache $G \leftarrow \varnothing$
Initialize $\mathcal{L}$ on Σ
$b_{\mathrm{MQ}} \leftarrow 0$; $b_{\mathrm{EQ}} \leftarrow 0$
while true do
  while PENDINGMQ($\mathcal{L}$) **do**
    $w \leftarrow$ NEXTMQ($\mathcal{L}$)
    if $w \in G$ **then**
      $\hat{y} \leftarrow G[w]$
    else if $w \in C$ **then**
      $\hat{y} \leftarrow C[w]$
    else
      if $b_{\mathrm{MQ}} = B_{\mathrm{MQ}}$ **then**
        return budget-exhaustion report and latest hypothesis, if available
      end if
      $\hat{y} \leftarrow$ LLMJUDGE(x, Σ, w)
      $C[w] \leftarrow \hat{y}$
      $b_{\mathrm{MQ}} \leftarrow b_{\mathrm{MQ}} + 1$
    end if
    ANSWERMQ($\mathcal{L}, w, \hat{y}$)
  end while
  $A \leftarrow$ HYPOTHESIS($\mathcal{L}$)
  if $b_{\mathrm{EQ}} = B_{\mathrm{EQ}}$ **then**
    return budget-exhaustion report and A
  end if
  $(\tau, o) \leftarrow$ EXECUTE(A)
  $b_{\mathrm{EQ}} \leftarrow b_{\mathrm{EQ}} + 1$
  $w_\tau \leftarrow \phi(\tau)$
  if MISMATCH(x, A, w_τ, o) **then**
    $y_\tau \leftarrow$ ENVLABEL(x, w_τ, o)
    $G[w_\tau] \leftarrow y_\tau$
    $\mathcal{L} \leftarrow$ REFINE($\mathcal{L}, w_\tau, y_\tau$)
  else if MISSINGPRED(x, Σ, τ, o) **then**
    $\Sigma \leftarrow$ EXPAND(x, Σ, τ, o)
    Reinitialize $\mathcal{L}$ on Σ
    Replay compatible labels from C and G
  else
    return A
  end if
end while

Persistent-noise reconciliation. For each queried word w, the cache stores the first Boolean membership judgment $\widetilde{y}(w)$ and returns that judgment on subsequent queries. CAPAL does not repeatedly sample the same word or aggregate multiple labels for that word. A counterexample label supplied through execution is instead stored as a gold override and takes precedence over the cached membership judgment.

CAPAL obtains robustness through a statistical class query over multiple suffixes. For prefixes u and v, let

$$\mathcal{E} = E_{\text{core}} \cup E'_{\text{pool}}, \qquad m = |\mathcal{E}|,$$

where E'_{pool} is a capped sample from the suffix pool. CAPAL computes the empirical disagreement rate

$$D(u, v) = \frac{1}{m} \left|\{e \in \mathcal{E} : \widetilde{y}(ue) \neq \widetilde{y}(ve)\}\right|$$

and declares u and v to represent the same Myhill–Nerode class iff

$$D(u, v) \leq p_0 + \tau, \qquad p_0 = 2\bar{\eta}(1 - \bar{\eta}),$$

$$\tau = \min\left\{\tau_{\max}, \sqrt{\frac{\ln(2/\alpha)}{2m}}\right\},$$

where $\bar{\eta}$ is a conservative upper bound on the membership-noise rate. Only DIFFERENT outcomes are cached. The cache key includes the prefix pair, suffix-set version, and decision threshold, so a comparison is recomputed when its evidence or threshold changes. During a comparison with m planned suffixes, CAPAL stops early and caches DIFFERENT when the accumulated disagreement count d_t satisfies

$$\frac{d_t}{m} > p_0 + \tau.$$

Counterexample update. CAPAL maintains a prefix-closed access set S, a core discriminator set E_{core}, a capped suffix pool E_{pool}, an observation table of cached membership judgments, and a discrimination tree over the current state classes. Given a labeled counterexample (w_τ, y_τ), CAPAL stores y_τ as a gold override and inserts the prefixes of w_τ into S.

CAPAL then applies Rivest–Schapire analysis, searching for the earliest factorization

$$w_\tau = uae$$

such that

$$\delta^A(A[u], a) \neq A[ua].$$

If such a factorization exists, the suffix e is added to E_{core} as a structural discriminator. If no split exists, the counterexample is treated as label-only: its gold label corrects the acceptance decision for the class it reaches, and the counterexample is recorded without immediately expanding the observation table. If the same label-only counterexample recurs, its suffixes are added to E_{core} to separate the incorrectly merged class.

When enforcing consistency, CAPAL uses the discrimination tree to select a discriminator from the lowest common ancestor of two distinct successor classes and adds the corresponding derived column to E_{core}. Any membership queries required by the resulting closedness or consistency repairs are issued lazily. CAPAL then rebuilds its state partition and hypothesis DFA; CEDAR does not directly patch an individual transition.

Budgets and termination. The membership and execution budgets in Algorithm 1 are experiment-level CEDAR limits rather than parameters required by CAPAL. Cache hits and gold-label lookups do not consume the membership-query budget; only a new call to LLMJUDGE does. Each execution of a hypothesis consumes one execution/EQ query.

Learning terminates when execution produces no recognized counterexample, when the experiment's success predicate is satisfied, or when either budget is exhausted. CAPAL ordinarily assumes a fixed alphabet. Alphabet expansion is therefore implemented as a CEDAR-level restart: the alphabet-dependent learner state is reinitialized, while cached membership judgments and gold counterexample labels for words that remain valid over the enlarged alphabet are replayed. This preserves previously acquired semantic evidence without reusing an invalid state partition or discrimination tree.

H Assertion-Baseline Statistics

Table 11 reports exact two-sided 95% binomial confidence intervals for the three binary outcomes in Table 2. The intervals are necessarily wide because each condition contains only five trials.

Method	Found diamond	Constraint compliant	Joint success
VOYAGER w/o A.	4/5 [0.284, 0.995]	0/5 [0.000, 0.522]	0/5 [0.000, 0.522]
VOYAGER w/ A.	4/5 [0.284, 0.995]	2/5 [0.053, 0.853]	0/5 [0.000, 0.522]
CODEACT w/o A.	3/5 [0.147, 0.947]	1/5 [0.005, 0.716]	0/5 [0.000, 0.522]
CODEACT w A.	0/5 [0.000, 0.522]	3/5 [0.147, 0.947]	0/5 [0.000, 0.522]
CEDAR	5/5 [0.478, 1.000]	5/5 [0.478, 1.000]	5/5 [0.478, 1.000]

Table 11: Observed success counts and exact two-sided 95% Clopper–Pearson confidence intervals for the five-trial sleep-at-night comparison. The intervals describe marginal binary outcomes and do not remove the uncertainty caused by the small sample. A.: `aseertions`.

I Goal-Completion Success Intervals

Table 12 reports exact two-sided 95% binomial confidence intervals for the success counts in Table 4. These intervals concern task success; they do not address the conditional prompting-iteration means.

Task	VOYAGER w/o S.L.	VOYAGER	CEDAR w/o S.L.	CEDAR
Wooden	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]
Iron	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]
Diamond	11/25 [0.244, 0.651]	17/25 [0.465, 0.851]	18/25 [0.506, 0.879]	23/25 [0.740, 0.990]
Lava	19/25 [0.549, 0.906]	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]	25/25 [0.863, 1.000]
Compass	17/25 [0.465, 0.851]	25/25 [0.863, 1.000]	9/25 [0.180, 0.575]	25/25 [0.863, 1.000]

Table 12: Task-success counts and exact two-sided 95% Clopper–Pearson confidence intervals over 25 trials per method–task condition. S.L. denotes a skill library.

J Automaton State Counts

The product of a skill DFA and a specification DFA has the standard worst-case upper bound $|Q_g||Q_c|$. Table 13 reports the component sizes and the number of reachable states in each composed automaton after unreachable product states are removed.

Metric	Diamond ∩ Sleep	Mine ∩ Night	Explore ∩ Biome
Skill DFA states	7	N/A	N/A
Specification DFA states	9	3	2
Reachable product states	17	3	2

Table 13: Component and reachable product-automaton sizes for the evaluated skill–constraint compositions. The worst-case product size is $|Q_g||Q_c|$; the reported product sizes exclude unreachable states.