LANGUAGE OPERATORS FOR GROUNDED REWARD INTERFACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Reward machines expose temporal reward structure to reinforcement learning (RL), but they are usually finite-state, grounded by hand in symbolic propositions, and learned or constructed one task at a time. These assumptions become restrictive when reward-relevant behavior requires memory beyond finite state, when symbolic task progress must be connected to continuous observations, or when an agent must solve a family of related task specifications rather than a single fixed task. We propose *language operators*: operators that learn maps from formal task specifications to grounded reward interfaces for RL. Given a specification, the interface predicts valid next events, task completion, and grounded targets from a trajectory prefix and the current observation. We learn these interfaces jointly across related tasks using supervision from formal monitors, and then keep the operator fixed during policy learning. The same parameters define reward-interface behavior for new specifications, allowing task descriptions to change without learning a separate reward model.

We instantiate this framework with recurrent and memory-augmented models and evaluate it on BabyAI, WaterWorld, and controlled counting and nesting tasks. Across controlled counting and nesting tasks, WaterWorld, and BabyAI, we compare language operators with learned task-conditioned machines, exact symbolic references with learned grounding, and task-conditioned or recurrent RL baselines. Controlled numerical experiments further demonstrate generalization to task parameters beyond the training range, showing how shared reward-interface learning can support both new task compositions and longer executions.

1 INTRODUCTION

Reward specification is a central bottleneck in reinforcement learning (RL). Sparse success signals are often easy to state but difficult to learn from, while dense shaping rewards are easier to optimize but difficult to design without changing the intended task. Reward machines (RMs)(Toro Icarte et al., 2018; 2022) address this problem by representing non-Markovian rewards as automata (Mealy machines) whose transitions emit rewards. They expose temporal task structure, support decomposition, and enable potential-based shaping. However, for agents expected to solve many related tasks, this raises a further question: can the process of constructing useful reward guidance itself be learned and reused?

Figure 1 illustrates the reuse problem. A BabyAI instruction may require two destinations in either order, one destination after another, or a sequence built from both patterns. The three examples induce different task structures: visiting the yellow box and door in A, visiting the green door before the red door in B, and visiting the green ball and a door before the red door in C. A conventional reward-machine workflow learns a separate machine for each new environment and relies on a symbolic event interface to track its state. As the family of missions grows, this task-by-task construction must be repeated.

The right side of the figure presents the alternative studied here. A single language operator is fitted across related specifications and then frozen. For each specification, it instantiates a different reward interface while reusing the same parameters. During interaction, the interface is queried with the trajectory prefix and current observation to identify reward-relevant progress, completion, and grounded targets. These query-time inputs are omitted from the schematic to keep the comparison

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

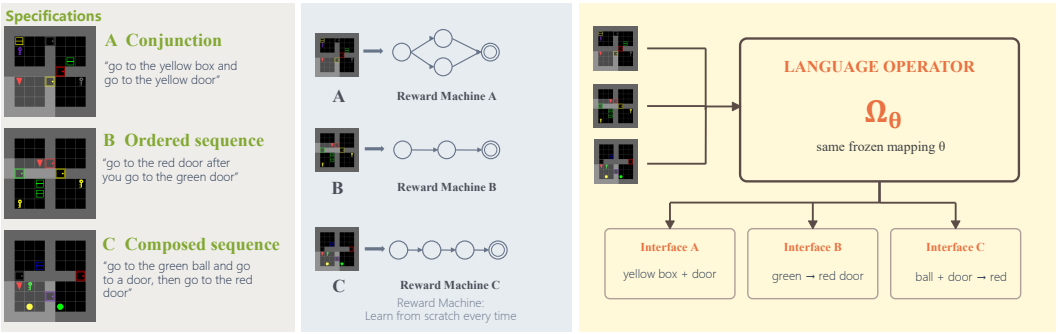


Figure 1: Task-specific reward machines and a shared language operator. Three BabyAI specifications illustrate conjunction (A), ordered sequence (B), and their composition (C). Learn a new reward machine from scratch in every new environment (middle). A single learned operator Ω_θ instead keeps the same parameters θ and instantiates a distinct grounded reward interface for each specification (right). The interface sketches show relevant targets and subgoal order; the drawing is conceptual rather than an empirical prediction.

focused on reuse across specifications. The question is whether a learned interface can capture this structure well enough to guide policy learning on new compositions.

We realize this idea with a *language operator*: a learned map conditioned on a formal task specification, formal constraints, a trajectory prefix, and the current object-centric observation. It predicts legal next events, grounded target objects, prefix validity, task completion, memory updates, and shaping potentials. The policy uses this structured interface for task context and dense feedback. All native-control experiments use PPO. We compare the learned interfaces with direct task-conditioned or recurrent policies and separately identified oracle-semantic references.

We train one operator over related tasks within an environment family. Its shared parameters learn how formal structure, prefix memory, and observation grounding interact, while the specification selects the task instance. This extends the RM perspective from a separately constructed finite monitor for each task to a learned reward-construction rule reused across specifications.

We use the term *language operator* for a shared learned map from a task specification to the reward-interface behavior queried during interaction. This is a specification-conditioned model, rather than a new neural-operator architecture. The central question is whether jointly learning temporal progress and grounded targets across a task family produces an interface that remains useful when its parameters are frozen and the task specification changes.

This perspective differs from both classical automata learning and generic recurrent RL. Automata-learning algorithms (such as $L^\star$ (Angluin, 1987), SAT-based DFA identification (Heule & Verwer, 2010), and weighted-automata learning (Balle & Mohri, 2015)) are powerful when the alphabet is known and examples are symbolic. They do not solve continuous grounding, dense reward generation, or transfer from one formal specification to another. Recurrent policies can in principle remember histories, but their memory is task-agnostic and does not expose the prefix-dependent task state or the next useful subgoal. Language operators occupy a middle ground: formal specifications and constraints provide structure, while the learned operator performs prefix-dependent prediction, object grounding, and dense reward construction.

A motivating deployment regime is one in which semantic annotations are available offline but an exact grounded interface is unavailable online. For example, a simulator may provide privileged event and task-progress labels during training, whereas the deployed agent receives only partial observations and interaction histories. An executable task monitor alone does not resolve this gap: it still requires identifying which symbolic events occurred. A language operator learns to infer task progress and grounded targets from the available history, then reuses this inference across specifications without querying the semantic teacher. Exact monitoring with learned grounding remains a direct alternative, which we include as a reference. Our experiments evaluate the usefulness of the learned replacement under monitor supervision; they do not establish that exact monitoring is infeasible in the evaluated benchmarks.

Contributions. We formulate reward construction as learning an operator from task specifications to grounded reward-interface functions and develop supervised realizations that remain fixed during downstream policy learning. We evaluate this framework on held-out BabyAI mission compositions, continuous-control WaterWorld, and parameterized Counter/Dyck tasks, separating semantic accuracy from native control performance.

We learn language operators from examples of task progress and grounding across a family of specifications. Formal monitors provide supervision for task progress, while observations provide the corresponding grounding information. Once learned, the operator is held fixed and its reward interface can be used to provide task context and construct dense feedback for an RL agent. We evaluate language operators on three complementary settings spanning counting and nesting tasks, continuous collection-and-recall control in WaterWorld, and compositional instruction following in BabyAI. We compare against task-conditioned and recurrent RL agents, learned task-conditioned reward models, and exact symbolic monitors with learned grounding. Across these settings, language operators provide effective reward interfaces for policy learning and generalize across task specifications, including held-out task compositions and task parameters beyond those observed during training. These results suggest that reward construction can itself be learned as a reusable, specification-conditioned mapping, rather than constructed independently for every task.

2 Related work

Instruction-conditioned reward learning. AGILE learns instruction-conditioned rewards from examples of goal states and demonstrates generalization to unseen instructions, including reuse of a frozen reward model for policy adaptation (Bahdanau et al., 2019). Our setting concerns structured, history-dependent interfaces for temporally extended tasks: in addition to completion, the interface predicts valid next events and grounded targets. We study these predictions under monitor supervision and evaluate their frozen reuse across task specifications.

Meta-RL and task-conditioned reward construction. Meta-RL studies learning across a distribution of tasks, typically by learning an initialization, recurrent policy, value function, or adaptation mechanism that supports rapid control on held-out tasks (Duan et al., 2016; Wang et al., 2016; Finn et al., 2017). Our setting shares the task-distribution viewpoint but changes the learned object. The language operator does not output a policy and does not perform inner-loop policy adaptation. Instead, it learns a task-conditioned reward-construction rule: a map from a formal specification, constraints, prefix history, and continuous observations to the reward-relevant signals supplied to RL.

Formal languages, weighted automata, and automata learning. Formal languages need not be Boolean. Weighted automata assign values from a semiring to words and can represent probabilities, costs, durations, and other quantitative behaviors (Droste et al., 2009). Classical grammatical inference learns automata from examples or queries, including $L^\star$ for regular languages (Angluin, 1987), SAT encodings of DFA identification (Heule & Verwer, 2010), and algorithms for weighted automata (Balle & Mohri, 2015). These methods are complementary to our goal. They usually assume a known symbolic alphabet and do not by themselves solve continuous grounding, reusable reward construction across task specifications, or dense reward feedback for exploration.

Reward machines and formal-language RL. Reward machines (RMs) expose reward-function structure as finite-state machines and support shaping, decomposition, and counterfactual updates (Toro Icarte et al., 2018; 2022). RMs can also be learned from experience in partially observable domains (Toro Icarte et al., 2019; 2023). Formal-language reward specification, including LTL and regular languages, can be translated into RMs (Camacho et al., 2019). LTL2Action instead conditions policies on temporal-logic instructions and learns to generalize to unseen formulas (Vaezipoor et al., 2021). Vision-based and neural variants of reward machines address symbol grounding by learning perception modules or differentiable automata (Camacho et al., 2021; Umili et al., 2024; Dewidar & Umili, 2025), and latent automaton models learn task structure from observation sequences (Zhan et al., 2025). Our distinction is that the object being generalized is not a policy and not a single per-task automaton, but a learned map from formal task specifications to grounded reward interfaces. An exact monitor combined with a learned grounder is a direct alternative when executable task semantics are available. We include this alternative to measure the performance gap incurred by replacing exact semantic execution with a learned interface. Our experiments evaluate this replacement under monitor supervision; they do not establish an advantage when task semantics lack an executable specification.

3 PRELIMINARIES

Tasks and histories. A task specification $x \in \mathcal{X}$ defines a formal task over an event alphabet Σ, together with static task information C_x. At time t, the agent has a public observation o_t and an interaction history h_t. For the formal semantics, $w_t \in \Sigma^*$ denotes the corresponding event prefix; an implementation can receive either this projected event prefix or the available interaction history from which task progress is inferred. Because reward-relevant progress can depend on history, it need not be determined by o_t alone.

Grounded reward interfaces. For a task x and event prefix w, let

$$V_x(w) = \{e \in \Sigma : we \in \mathrm{Pref}_{C_x}(L_x)\}$$

denote the valid next events, and let

$$A_x(w) = \mathbf{1}\{w \in L_x\}$$

denote task completion. Let $\mathcal{T}(o)$ denote the set of target candidates available in observation o. Grounding is represented by

$$G_x(o, w) \in \Delta(\mathcal{T}(o) \cup \{\varnothing\}),$$

where $\varnothing$ denotes that no target should be selected.

We write the grounded reward interface abstractly as

$$\mathcal{I}^\star_x(w, o) = \big(V_x(w), A_x(w), G_x(o, w), \dots\big),$$

where additional quantities such as prefix validity or shaping potentials may be included when required by the domain. Full definitions are given in Appendix A.

4 METHOD

4.1 LANGUAGE OPERATORS AND SEMANTIC INVARIANCE

A language operator is a learned map from task specifications to grounded reward-interface functions:

$$\Omega_\theta : \mathcal{X} \to \mathcal{F}, \qquad x \mapsto \widehat{\mathcal{I}}_{\theta,x}, \tag{1}$$

where $\mathcal{F}$ is a space of reward-interface functions and

$$\widehat{\mathcal{I}}_{\theta,x} : (h, o) \mapsto \widehat{\mathcal{I}}_{\theta,x}(h, o).$$

Equivalently, we write

$$\Omega_\theta(x, h, o) := [\Omega_\theta(x)](h, o).$$

We call a language operator *semantically invariant* over a task family when the same learned operator Ω_θ is reused as the task semantics change, without fitting new parameters or a separate task-specific model. Different specifications generally induce different reward-interface functions $\widehat{\mathcal{I}}_{\theta,x}$; invariance refers to the shared operator, not to identical outputs. For example, "pick up the apple" and "pick up the key" induce different grounded interfaces, while different values of n induce different progress functions for counting or nesting tasks, but the same Ω_θ maps each specification to its corresponding interface.

4.2 LEARNING AND USING THE OPERATOR

Given a task specification x, interaction history h_t, and current observation o_t, the operator predicts a grounded reward interface

$$\widehat{\mathcal{I}}_{\theta,x}(h_t, o_t) = \Omega_\theta(x, h_t, o_t),$$

which may contain valid next events, task completion, grounded targets, and domain-specific auxiliary quantities. We fit one operator per environment family across training specifications. Executable formal monitors provide progress-related supervision during offline data generation, while scene-dependent labels supervise grounding. The domain-specific objective is

$$\mathcal{L}_d(\theta) = \sum_{k \in \mathcal{H}_d} \lambda_{d,k} \mathcal{L}_{d,k}(\theta), \tag{2}$$

where $\mathcal{H}_d$ contains the supervised interface components used in domain d.

After fitting, Ω_θ is frozen. During policy learning and evaluation, it predicts interface quantities from the task specification, available history, and observation without receiving the teacher's prefix-dependent semantic outputs. The resulting interface can be supplied to the policy as task context or used to construct dense guidance signals. BabyAI uses a recurrent history encoder, Counter/Dyck combines learned local control with exact generic memory and arithmetic operations, and WaterWorld uses a separately fitted progress-and-grounding interface. Full architectures, supervised objectives, and interface-to-control mappings are given in Appendices B–D.

5 Empirical Results

We first ask whether a frozen language operator recovers reward-relevant structure for specifications outside its training range. We then ask whether that interface helps PPO learn in three RL benchmarks: BabyAI, WaterWorld, and native colored-Dyck control. The benchmarks test different forms of reuse: held-out instruction compositions, continuous collection-and-recall control, and longer numerical tasks. Operators are fitted separately in each domain; these experiments do not transfer one set of weights across domains.

5.1 Experimental setup

BabyAI uses 200 training, 24 development, and 24 held-out canonical mission identities, with 16 held-out layouts per mission. WaterWorld requires eight collection and eight recall events at a fixed task size. Counter requires $X^nY^nZ^n$GOAL, and colored Dyck requires n matched pairs with last-in-first-out closing. In the numerical study, the formal operator is trained on arguments 1–12 and frozen for tests at 13–16. Native Dyck policies train only through count 4.

We report semantic correctness separately from native task completion and training return. The external-domain comparisons include LO-guided PPO, context-only PPO with the learned interface but no added guidance reward, and direct task-conditioned or recurrent PPO. An exact-semantic reference uses the same learned grounding and target pathway. These arms do not have equal offline supervision, and the recurrent WaterWorld control also differs in architecture. The numerical semantic comparison includes a learned task-conditioned machine and two exact references. Full protocols, training objectives, and comparison contracts appear in Appendices E and C.

5.2 Does LO learn reward-relevant structure?

Offline operator fitting. LO is trained before PPO on task-conditioned examples labeled by an executable semantic monitor and scene-dependent grounding targets. BabyAI trains the interface jointly, whereas Dyck and WaterWorld fit their grounding components separately. In the main experiments, the resulting interface is frozen before policy learning.

Figure 2 reports an independent offline training sweep using the registered recipes. The sweep includes five Dyck fits with 1,500 local-controller updates each and three BabyAI fits with 6,000 joint updates each. Optimization loss is recorded at every update, while development semantic accuracy is evaluated every 25 updates for Dyck and every 50 updates for BabyAI. At the final recorded update, BabyAI reaches a mean stage-balanced progress-set accuracy of 0.775. All five Dyck fits reach 1.0 all-head exact accuracy on the fixed in-range development probe. Because the two domains use different labels and objectives, their loss values are not directly comparable. This sweep contains neither RL training nor TEST evaluation, so these curves diagnose offline interface fitting rather than policy learning. Generalization to unseen specifications is evaluated separately by the frozen semantic tests below.

To test whether this fitted operator recovers task structure, a correct full trace requires every scored progress prediction along a history to be correct. A parameter-edit probe changes the public numerical argument while holding the history and learned weights fixed. Thus, these tests ask whether the same fitted map executes a new specification rather than memorizing one reward sequence.

Table 1 shows that the frozen LO is correct on all 280 Counter and 480 Dyck held-out full traces, and on all 30 Counter and 45 Dyck parameter edits, pooled over five formal fits per domain. The learned

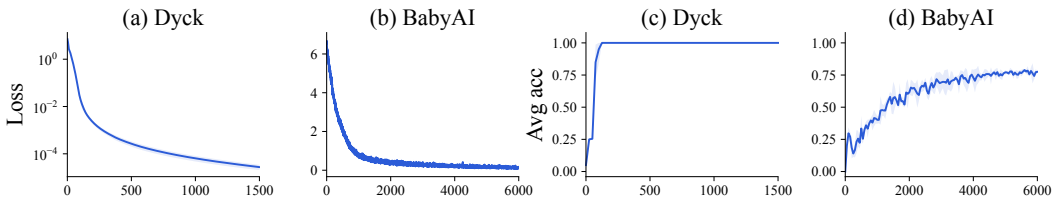


Figure 2: Offline LO fitting on an independent dense rerun. From left to right: Dyck loss, Dyck development semantic accuracy, BabyAI loss, and BabyAI development semantic accuracy. Loss is recorded before each optimizer step, and accuracy is evaluated after the step every 25 updates for Dyck and every 50 updates for BabyAI. Dyck reports all-head exact accuracy on a fixed in-range probe across five fits; BabyAI reports stage-balanced progress-set exact accuracy across three fits. Lines show means and shaded bands show sample SD, with no smoothing. The Dyck loss axis is logarithmic. Update-zero accuracy is measured at random initialization, whereas loss begins at the first update. The Dyck panels report the formal component. This sweep contains no RL training or TEST evaluation and is separate from the held-out semantic results reported below.

Table 1: Frozen numerical semantic generalization to counts 13–16 after training on 1–12. Entries are correct/evaluated full traces or parameter edits, pooled over five formal fits per domain. The exact rows are reference systems; the learned machine differs in capacity and supervision.

Method	Counter full traces	Counter edits	Dyck full traces	Dyck edits
Parameterized LO	280/280	30/30	480/480	45/45
Learned task machine	30/280	0/30	16/480	0/45
Exact known-structure recompiled	280/280	30/30	480/480	45/45
Exact logic + learned grounder	280/280	30/30	480/480	45/45

task machine is correct on 30/280 and 16/480 traces and on none of the edits. Exact references pass all measured tests. LO uses learned local control with exact generic memory and arithmetic; the learned machine has different capacity and supervision. The result therefore supports generalization of this implemented reward interface over the measured counts, not an architecture-only or unbounded-correctness claim. Per-fit and per-count results are in Appendix H.

Component-level semantic scores and specification interventions are reported in Appendix F.

5.3 RL BENCHMARK 1: BABYAI COMPOSITIONS

BabyAI tests whether the frozen interface can guide policies on new compositions of familiar instruction primitives. Each policy trains for 3,145,728 interactions; final fixed-budget policies are evaluated on the same 384 held-out layouts. Figure 3 shows the training comparison. At roughly one million interactions, mean native return is 0.800 for LO-guided PPO, 0.669 for context-only PPO, and 0.265 for direct task-conditioned PPO. At the end of training, the respective returns are 0.917 ± 0.006, 0.910 ± 0.011, and 0.708 ± 0.044; the exact-semantic reference reaches 0.932 ± 0.006. The learned interface therefore provides a large early advantage, while the gap between guidance and context-only control narrows during training.

Table 2 gives the separate held-out completion test. LO-guided PPO completes $86.55 \pm 2.25\%$ of missions, context-only PPO $84.33 \pm 2.11\%$, and direct PPO $57.20 \pm 9.50\%$. The exact-semantic reference reaches $94.44 \pm 0.49\%$. Each value is a mean and sample SD over three interface/grounder bundles, each with two policy seeds. The oracle gap leaves room to improve the learned interface, although this comparison does not isolate the source of that gap. Mission-level and paired-bundle results are in Appendix F.

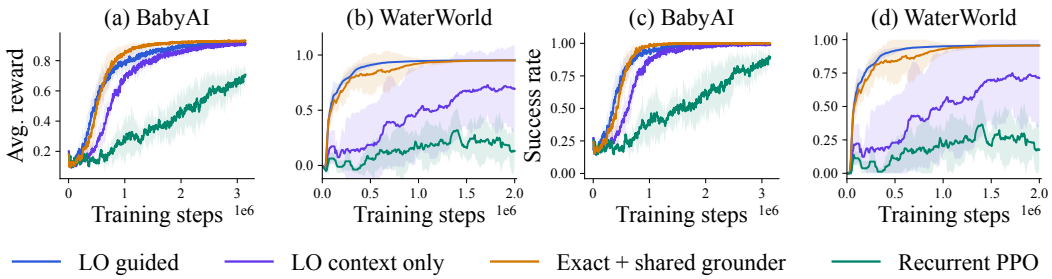


Figure 3: Training curves for the first two RL benchmarks: BabyAI native average reward and success rate (left), and WaterWorld normalized native reward and observed success rate (right). BabyAI uses the last at most 100 completed episodes at each 2,048-interaction update, averaging three fitted bundles with two policy seeds each. WaterWorld uses trailing ten 8,000-interaction rollouts, averaging five policy seeds per arm. Its reward is $(S-T/10^5)/(S+F+8)$ and its success rate is $S/(S+F+8)$, including eight still-running episodes in each denominator. Bands show sample SD. The reward panels have different estimands.

Table 2: Native completion (%) in the earlier three-bundle confirmation cohorts. Values are mean ± sample SD across bundle averages of two policy seeds. Direct control is task-conditioned PPO in BabyAI and recurrent PPO in WaterWorld. The exact-semantic reference shares learned grounding. WaterWorld uses deterministic actions at fixed task size; its newer learning curves use a separate five-seed cohort.

Method	BabyAI held-out completion (%)	WaterWorld deterministic completion (%)
LO-guided	86.55 ± 2.25	99.48 ± 0.90
LO context-only	84.33 ± 2.11	98.96 ± 1.19
Direct PPO control*	57.20 ± 9.50	0.00 ± 0.00
Exact + shared learned grounder	94.44 ± 0.49	99.48 ± 0.90

5.4 RL BENCHMARK 2: WATERWORLD CONTINUOUS CONTROL

WaterWorld tests the same interface design in a continuous collection-and- recall task, using a newly fitted operator. Figure 3 shows all 250 recorded rollouts per policy in the completed two-million-interaction cohort, with five seeds for each of four arms. The curves use native outcomes only, excluding method-specific intermediate rewards. Their first recorded point is at 8,000 interactions, where all arms have normalized reward -0.01. A trailing window includes episodes still running at its end; the plotted statistic is therefore distinct from completed-episode return or a zero-step policy evaluation.

At the final paired development checkpoint, LO-guided and the exact-semantic reference each complete 60/60 episodes with sampled and deterministic actions. Context-only completes 49/60 under either mode; recurrent PPO completes 26/60 sampled and 20/60 deterministic episodes. These are development measurements. In the separate earlier confirmation cohort, the deterministic terminal completion values are $99.48 \pm 0.90\%$, $98.96 \pm 1.19\%$, and $0.00 \pm 0.00\%$ for LO-guided, context-only, and recurrent PPO, respectively (Table 2). The training recipe and cohort differ from that study, so their results should not be read as a controlled before/after comparison. Since the task size is fixed and training can encounter all 70 balanced parity orders, this experiment establishes continuous-control utility rather than count extrapolation.

5.5 RL BENCHMARK 3: DYCK CONTROL AT LARGER COUNTS

Native Dyck tests whether a policy can use a frozen learned interface on larger task arguments. Two LO policies train through count 4 with one shared formal/grounder fit trained through count 12. We then evaluate their fixed weights at counts 13–16. Table 3 reports both completion and whether an episode actually reaches the requested maximum nesting depth. LO completes 507/512 episodes

Table 3: Frozen native Dyck results at held-out counts. Each entry pools two policy seeds and gives correct episodes out of 128. The LO formal interface was trained through count 12 and its policies through count 4. "Depth" means the maximum attained nesting depth equals the requested count. The exact-target diagnostic has privileged target access.

	LO		Exact target	
Requested count	Completed	Depth	Completed	Depth
13	128/128	128/128	125/128	128/128
14	127/128	127/128	125/128	128/128
15	127/128	126/128	124/128	126/128
16	125/128	124/128	123/128	126/128
Total	507/512	505/512	497/512	508/512

overall, compared with 497/512 for the privileged exact-target diagnostic. At count 16, the two LO policies complete 63/64 and 62/64 episodes, and each reaches depth 16 in 62/64.

The actor receives interface-derived target geometry, not direct semantic channels. Its reward includes local-progress and non-potential event guidance, so the result establishes long execution under this control construction, not policy invariance of the complete shaped reward. Native histories are predominantly single-color; the mixed-color semantic test is the separate full-trace evaluation above. Per-policy completion, depth intersections, and within-curriculum learning curves appear in Appendix H.

6 Conclusion

We introduced language operators as learned maps from task specifications to grounded reward-interface functions. A fixed operator can instantiate useful interfaces for held-out mission compositions and larger numerical task arguments. Across BabyAI, WaterWorld, and controlled counting and nesting studies, these interfaces connect task progress to the information and guidance used by an RL agent. Context-only results show that their usefulness extends beyond added reward shaping. The experiments support learning reward construction across task families, while leaving exact semantics and broader generalization as directions for further study.

AI Use Statement

We used AI-assisted tools for literature search and related-work discovery. AI tools were not used to autonomously determine the scientific conclusions of this work. The authors reviewed and verified the final manuscript, experimental results, and cited references, and take full responsibility for the content of the paper.

Reproducibility Statement

To support reproducibility, we provide the implementation of the proposed method together with the code required to run the experiments. We also provide the benchmarks used in our evaluation, or instructions for obtaining publicly available benchmarks when redistribution is not permitted. The experimental configuration, including model architectures, training procedures, hyperparameters, random seeds, number of independent runs, evaluation protocols, and statistical procedures used to aggregate results and quantify uncertainty, is documented in the accompanying code and supplementary material. These materials include the configurations and scripts needed to reproduce the reported experiments and results.

References

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

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward, 2019. URL https://arxiv.org/abs/1806.01946.

Borja Balle and Mehryar Mohri. Learning weighted automata. In *Algebraic Informatics*, volume 9270 of *Lecture Notes in Computer Science*, pp. 1–21. Springer, 2015.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard A. Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 6065–6073, 2019.

Alberto Camacho, Jacob Varley, Andy Zeng, Deepali Jain, Atil Iscen, and Dmitry Kalashnikov. Reward machines for vision-based robotic manipulation. In *IEEE International Conference on Robotics and Automation*, pp. 14284–14290, 2021.

Hazem Dewidar and Elena Umili. Fully learnable neural reward machines. *arXiv preprint arXiv:2509.19017*, 2025.

Manfred Droste, Werner Kuich, and Heiko Vogler (eds.). *Handbook of Weighted Automata*. Springer, 2009.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

Marijn J. H. Heule and Sicco E. Verwer. Exact DFA identification using SAT solvers. In *Grammatical Inference: Theoretical Results and Applications*, volume 6339 of *Lecture Notes in Computer Science*, pp. 66–79. Springer, 2010.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pp. 278–287, 1999.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2107–2116, 2018.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Q. Klassen, Richard Valenzano, Margarita P. Castro, and Sheila A. McIlraith. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, Margarita P. Castro, Ethan Waldie, and Sheila A. McIlraith. Learning reward machines: A study in partially observable reinforcement learning. *Artificial Intelligence*, 323:103989, 2023.

Elena Umili, Francesco Argenziano, and Roberto Capobianco. Neural reward machines. *arXiv preprint arXiv:2408.08677*, 2024.

Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. LTL2Action: Generalizing LTL instructions for multi-task RL. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10497–10508, 2021.

Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Weixiao Zhan, Qiyue Dong, Eduardo Sebastián, and Nikolay Atanasov. LATMOS: Latent automaton task model from observation sequences. *arXiv preprint arXiv:2503.08090*, 2025.

A FORMAL DETAILS FOR GROUNDED REWARD INTERFACES

Task specifications. A task specification

$$x = (x_1, \ldots, x_{\ell_x}) \in \mathcal{X}$$

denotes a formal task language $L_x \subseteq \Sigma^*$ together with static task information C_x. The task information C_x contains quantities known before execution and may include the task family, event support, object or symbol arguments, numerical parameters, forbidden events, or hazard constraints. It does not contain prefix-dependent oracle information such as the current valid-next set, parser phase, completion state, or outputs of the formal monitor.

Interaction histories and event projection. Let the environment have state s_t, action a_t, public observation o_t, and interaction history

$$h_t = (o_0, a_0, \ldots, a_{t-1}, o_t).$$

For task families with an explicit event alphabet, environment interactions induce events through

$$E : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Sigma \cup \{\varnothing\},$$

where $\varnothing$ denotes an interaction that produces no task-relevant event. Deleting null events gives the formal event prefix

$$w_t = e_1 e_2 \cdots e_k \in \Sigma^*.$$

The formal semantics are defined using w_t, while a learned realization may consume either w_t or the richer public history h_t.

Prefix-dependent task quantities. Define

$$\mathrm{Pref}_{C_x}(L_x) = \{v \in \Sigma^* : \exists u \in \Sigma^* \text{ such that } vu \in L_x \text{ and } v \text{ satisfies } C_x\}.$$

For a prefix w,

$$V_x(w) = \{e \in \Sigma : we \in \mathrm{Pref}_{C_x}(L_x)\}, \tag{3}$$
$$A_x(w) = \mathbf{1}\{w \in L_x\}, \tag{4}$$
$$B_x(w) = \mathbf{1}\{w \in \mathrm{Pref}_{C_x}(L_x)\}. \tag{5}$$

These quantities denote valid next events, task completion, and prefix validity, respectively.

Grounding and target selection. For an observation o, let $\mathcal{T}(o)$ denote the available target candidates. The grounding component is

$$G_x(o, w) \in \Delta(\mathcal{T}(o) \cup \{\varnothing\}),$$

where $\varnothing$ denotes that no target should be selected. The target space is domain dependent: for example, it may contain object indices in an object-centric domain or visible target locations in a grid observation.

Full interface. A domain may additionally expose prefix validity, a shaping potential $\Phi_x(w, o)$, or auxiliary task-state quantities. A general interface can therefore be written as

$$\mathcal{I}_x^\star(w, o) = (V_x(w), G_x(o, w), A_x(w), B_x(w), \Phi_x(w, o)),$$

with unused components omitted by a particular realization. Auxiliary quantities used only to train an internal representation are not required components of the abstract reward interface.

B Domain-specific operator implementations

B.1 BabyAI

The BabyAI realization receives the mission specification, the public interaction history, and the current partial grid observation. Unlike the projected event representation used in the formal definitions, the history encoder operates on the available public history, including repeated and non-progress interactions.

The history is encoded by an ordinary GRU. Mission and observation encoders provide the specification and scene representations used by the interface predictor. The supervised interface quantities include task progress, task completion, an auxiliary stage representation, object grounding, and target selection. The same learned operator parameters are shared across mission specifications.

No exact semantic monitor output is provided to the learned operator at deployment. The GRU is a domain-specific realization of the history encoder rather than a defining component of language operators.

B.2 Parameterized Counter and Dyck

The controlled Counter/Dyck realization uses a continuous 2D point-mass environment with object-centric observations. The public observation has the form

$$o_t = (p_t, \mathcal{O}_t), \qquad \mathcal{O}_t = \{z_t^j\}_{j=1}^{N_t},$$

where p_t is the normalized agent position and $\mathcal{O}_t$ is the set of currently available objects. Object features include continuous coordinates, observable color features, object-kind indicators, and an active-object mask. Internal symbolic object identities are reserved for the environment monitor and exact reference methods; the learned grounding pathway operates on the observable features.

B.3 WaterWorld

WaterWorld uses a separately fitted realization of the language-operator framework for continuous collection-and-recall control. The operator receives the task specification, public interaction history, and the current continuous scene representation. Its history and scene representations are combined to predict task-progress and grounded-target information used by the control interface.

The learned parameters are shared across task specifications within the WaterWorld family but are fitted separately from the BabyAI and Counter/Dyck operators. The exact observation dimensions, encoder sizes, and grounding architecture are reported in the reproducibility tables.

C Domain-specific supervised objectives

Language operators are trained offline from task-conditioned examples. Where executable task semantics are available, a formal monitor is queried only during data generation to provide semantic supervision. The learned operator itself does not query this monitor after training.

For each domain d, let $\mathcal{H}_d$ denote the active supervised interface components. The training objective is

$$\mathcal{L}_d(\theta) = \sum_{k \in \mathcal{H}_d} \lambda_{d,k} \mathcal{L}_{d,k}(\theta).$$

BabyAI. The BabyAI objective supervises progress, completion, auxiliary stage, object-grounding, and target-selection quantities. Classification outputs use the corresponding cross-entropy or binary cross-entropy losses.

Counter and Dyck. The parameterized formal-language realizations supervise the learned local-control decisions. Generic arithmetic, register, and stack operations are executed exactly and are therefore not treated as learned semantic predictions.

WaterWorld. WaterWorld uses the task-progress and grounding objectives associated with its registered interface realization.

The complete active-head sets, objective weights, optimizer settings, and training schedules are reported in the reproducibility tables.

D Domain-specific control interfaces

D.1 BabyAI

The learned interface supplies task context together with a grounded target. When interface-derived shaping is enabled, the selected target $j_t^\star$ induces the geometric potential

$$\Phi_\theta(h_t, o_t) = -d(p_t, p_{j_t^\star}),$$

and the corresponding potential-difference term is

$$F_t = \gamma\Phi_\theta(h_{t+1}, o_{t+1}) - \Phi_\theta(h_t, o_t).$$

This component has the standard potential-based shaping form (Ng et al., 1999).

The context-only condition uses the same frozen learned interface as policy context while omitting this additional shaping term.

D.2 Native Dyck control

In the native Dyck realization, the policy receives target geometry derived from the frozen interface rather than direct semantic prediction channels. The interface additionally contributes local-progress guidance and an event bonus.

Because the event bonus is not a potential difference, the complete reward in this setting is not purely potential-based and does not inherit the standard policy-invariance guarantee for potential shaping alone.

D.3 WaterWorld

The WaterWorld realization supplies task-progress and grounded-target information to the continuous-action policy and reward construction. The complete reward coefficients and policy-input contract are reported in the corresponding reproducibility table.

D.4 Potential-based shaping on augmented reward states

For a history-dependent task, potential shaping can be interpreted on an augmented reward state containing the environment state, task specification, and task history. For example, when w_t is a sufficient task-history representation, define

$$\tilde{s}_t = (s_t, w_t, x).$$

A potential-difference term

$$F(\tilde{s}_t, a_t, \tilde{s}_{t+1}) = \gamma\Phi(\tilde{s}_{t+1}) - \Phi(\tilde{s}_t)$$

has the standard potential-based form on this augmented state (Ng et al., 1999). This statement applies to the potential component only; additional guidance terms such as the native-Dyck event bonus are treated separately.

E Reproducibility details

E.1 Study protocols and comparison inputs

Table 4 lists the training, development, and final-evaluation populations and the replication units for each study. The three external-domain interface/grounder fits each have two nested policy seeds per

Table 4: Training and evaluation protocols by study. Numerical semantic fits and native-control fits are counted separately. Development diagnostics are not pooled with final confirmation.

Study	TRAIN / DEV boundary	Final panel	Replication
BabyAI confirmation	200 canonical TRAIN missions; shared offline corpus; 24 DEV missions; interface selection only; separate same-mission layouts	24 disjoint TEST signatures; same 384 layouts/policy	3 interface/grounder fits x 2 policies x 4 arms; 24 policies
WaterWorld confirmation	Fixed 8 collection + 8 recall task; policy can encounter all 70 balanced parity orders; 50/20 formal word split; policy checks 1M and 2M	64 deterministic + 16 sampled fresh layouts/ policy; fixed size	3 bundles x 2 nested policy seeds x 4 arms
Counter/Dyck semantics	Counts 1–12; offline semantic and local executor targets differ by arm; Separate development examples for interface selection	Frozen counts 13–16; semantic traces/edits only	5 independent fits/ domain/method; 4 methods
Dyck native	Policy through n=4; formal model through n=12; Read-only n=1–8 and n=9–12 probes; numerical range previously inspected	Fresh frozen n=13–16; 64 layouts/count/policy	1 formal/grounder fit; 2 policy seeds/method; privileged exact-target comparator

Table 5: BabyAI comparison inputs, supervision, and control interfaces. Exact semantics uses the shared learned grounder. Context-only retains learned memory and targets; the direct policy receives no offline interface supervision.

Method	Inputs / semantics	Offline supervision	Reward
LO-guided	Public view, mission, raw history; frozen interface and learned target context; learned GRU	shared learned grounder plus offline formal teacher	native + registered potential shaping
LO context-only	Public view, mission, raw history; frozen interface and learned target context; learned GRU	shared learned grounder plus offline formal teacher	native only
Task-conditioned PPO	Public view, mission, raw history; zero interface context; none	no offline interface labels	native only
Exact + shared learned grounder	Public view, mission, raw history; frozen interface and learned target context; exact reference	shared learned grounder; privileged exact semantics	native + registered potential shaping

arm. The five-fit numerical semantic study is distinct from the one-interface-fit native Dyck study. We report individual policies and bundle means rather than treating evaluation episodes as independent training runs.

The comparison tables distinguish public inputs, learned and exact components, and offline supervision. The BabyAI direct control uses the same policy backbone and actor capacity as its interface arms but does not receive their offline labels. WaterWorld's recurrent control uses a different policy architecture. Context-only retains the learned interface and memory while removing its added guidance reward. The exact/shared-grounder references replace semantic predictions; native Dyck's exact-target diagnostic has additional privileged target access.

Table 6: WaterWorld comparison inputs and control interfaces. The three interface arms share the learned grounding/target pathway; recurrent PPO uses its own policy architecture and native reward.

Method	Public/context inputs	Reward	Offline supervision
LO-guided	["native public vector", "learned phase/top probabilities", "learned target geometry/confidence"]	-1 + learned intermediate progress + native terminal success	Offline primitive/control/ reward/readout labels on fixed 50 TRAIN words; 512000 synthetic grounding scenes per independently fitted bundle
LO context-only	identical to LO-guided	-1 + native terminal success; exactly zero intermediate reward	Identical frozen bundle to paired LO-guided
Recurrent no-LO PPO	["native public vector", "ordered current public event batch including repeats", "public fixed task descriptor [8 collections, 2 recall object roles]"]	-1 + native terminal success; exactly zero intermediate reward	No primitive, formal state, reward-class or grounder supervision
Exact + shared learned grounder	["native public vector", "exact phase/top one-hot", "same learned target geometry/confidence path"]	-1 + exact intermediate progress + native terminal success	Same fitted grounder as paired LO; exact formal structure is privileged

Table 7: Numerical semantic-comparison implementations. The learned machine and LO have different capacities and supervision. Listed registry defaults are not new checkpoint measurements.

Method	Structure / task input	Registered capacity contract	Offline supervision
Parameterized LO	lossless public n and bindings; full replay after edits	six learned control states; stack16; prefix128; width checkpoint-declared	common semantic targets plus offline local state
Learned task machine	random_learned_task_ conditioned_tables; lossless public n and bindings through learned conditioning	fixed K=64 default; counter v1 eight lexical roles; color-preserving global events; prefix128	common semantic targets; no local executor targets and no exact table initialization
Exact NRM (recompiled)	exact_compiled_public_ task; exact compiler receives each public task argument	task-dependent exact state count; native Dyck cap1000000 states/ 320MB per machine/ 1GB cache	exact formal structure; learned grounding only; clean temporal events in controlled semantic regime
Exact logic + learned grounder	analytic_public_reference; analytic parameterized reference	generic analytic reference; prefix128; stack16	analytic semantics; learned grounding only

E.2 Architecture, optimization, and control settings

The following tables give the saved architecture, history capacity, optimizer, reward, and selection settings. Numerical arguments, stack capacities, and attained depths are different quantities. The formal implementation uses learned local control with exact generic operations; it is not the earlier soft-stack implementation. Fields unavailable in the supplied configuration records remain N/A rather than being filled from another domain's settings. The accompanying CSVs identify source records and configuration fields.

Table 8: BabyAI implementation and optimization settings (1/3). Values belong to this study; unavailable fields are shown as N/A.

Saved field	Value
interface_count	3
policy_count	24
policy_budget	3145728
policy_selection	terminal_fixed_budget
preprocessing	BabyAIPublic raw events, max_history=4096; FamilySpatialPolicy IEEE v2
native_horizons	unchanged native absorbing deadlines; no horizon extension
action_selection	greedy argmax
interface	lo_babyai_gru_v1
env_id	BabyAI-GoToSeqS5R2-v0
max_history	4096
interface_width	64
interface_updates	6000
interface_batch_size	64
interface_learning_rate	3×10^{-4}

Table 9: BabyAI implementation and optimization settings (2/3). Values belong to this study; unavailable fields are shown as N/A.

Saved field	Value
interface_selection	DEV stage-balanced progress exact + .25 visible-target accuracy + .2 grounding accuracy; earliest tie
policy_width	128
num_envs	32
rollout_steps	64
epochs	4
env_minibatches	4
learning_rate	3×10^{-4}
adam_eps	10^{-8}
gamma	0.99
gae_lambda	0.95
clip	0.20
entropy	0.01
value_coef	0.50
max_grad_norm	0.50

Table 10: BabyAI implementation and optimization settings (3/3). Values belong to this study; unavailable fields are shown as N/A.

Saved field	Value
reward	native + .1*(.99*Phi(post)*(1-done)-Phi(pre)); native mission deadline absorbing; no bonus
train_missions	200
dev_missions	24
test_missions	24
formal loss weights (not in supplied config fields)	N/A

Table 11: WaterWorld implementation and optimization settings (1/2). Values belong to this study; unavailable fields are shown as N/A. WaterWorld template entries describe the retained architecture configuration, not additional confirmation fits.

Saved field	Value
steps	3000000
n_envs	5
horizon	10000
learning_rate	3×10^{-6}
recurrent_hidden_size	64
selection	fixed terminal checkpoint; no score-dependent stop, retry, or selection
n_steps	10000
batch_size	3000
n_epochs	30
gamma	0.99
gae_lambda	0.90
clip_range	0.15
clip_range_vf	1000
vf_coef	0.50

Table 12: WaterWorld implementation and optimization settings (2/2). Values belong to this study; unavailable fields are shown as N/A. WaterWorld template entries describe the retained architecture configuration, not additional confirmation fits.

Saved field	Value
target_kl	0.30
formal_updates_per_attempt	2000
grounder_updates_per_attempt	2000
stack_capacity	16
prefix_capacity	120000
object_capacity	10
formal loss weights (not in supplied config fields)	N/A

Table 13: Native Dyck implementation and optimization settings (1/3). Values belong to this study; unavailable fields are shown as N/A.

Saved field	Value
policy_training_counts	[1, 2, 3, 4]
formal_training_counts	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
counts	[13, 14, 15, 16]
policy_seeds	[0, 1]
formal_fits	1
horizon	250
selection	fixed original terminal policies
prior_13_16_range_inspected	true
backend	sb3
reward_rule	local_progress
gamma	0.99
learning_rate	10^{-3}
alpha	0.10
beta	0.10

Table 14: Native Dyck implementation and optimization settings (2/3). Values belong to this study; unavailable fields are shown as N/A.

Saved field	Value
width	128
num_envs	16
rollout_steps	128
epochs	4
minibatches	4
entropy_coef	0.01
value_coef	0.50
max_grad_norm	0.50
adam_eps	10^{-8}
stage_counts	[1, 2, 4]
stage_steps	524288
warmup_steps	65536
inputs	geometry context28; first24 channels zero
history_capacity	128

Table 15: Native Dyck implementation and optimization settings (3/3). Values belong to this study; unavailable fields are shown as N/A.

Saved field	Value
history_compression	none
phase_continuation	in-process policy and Adam; fresh native resets; no GAE splice
formal loss weights (not in supplied config fields)	N/A

E.3 SUPERVISION AND COMPUTATIONAL EXPOSURE

Each BabyAI interface fit uses the same corpus of 217,681 training and 3,596 development teacher queries, sampling 384,000 training examples across 6,000 updates. These are shared corpus sizes, not three disjoint corpora. The saved formal and grounding modules contain 144,394 and 45,980 parameters, respectively. Table 16 gives the per-fit checkpoint selection and exposure. Teacher actions are not PPO supervision.

Table 17 separates policy-training and evaluation interactions. The 24 BabyAI policies use 75,497,472 training interactions. Its broader charged accounting is 76,873,562 interactions, including development, TEST, teacher-evaluation, and correctness checks; inventory resets are recorded separately. WaterWorld uses 72,000,000 policy training interactions and 82,260,735 total charged interactions, of which 10,260,735 are outside policy training. Totals and subtotals are not added together, and interaction counts are not presented as measured wall-clock time.

Table 16: BabyAI interface supervision and saved parameter counts. The three fits use a shared teacher corpus; sampled training exposure and selected updates are listed per fit. No teacher action labels train PPO.

Fit	TRAIN / DEV teacher queries	Sampled examples / updates	Selected update	Formal / grounder parameters
100	217681 / 3596	384000 / 6000	5000	144394 / 45980
101	217681 / 3596	384000 / 6000	4000	144394 / 45980
102	217681 / 3596	384000 / 6000	6000	144394 / 45980

Table 17: Policy and campaign interaction accounting. BabyAI rows cover policy training and evaluation, not every auxiliary cost. WaterWorld total and beyond-training rows include evaluation/checking costs and must not be added to the training subtotal. Reset-only inventory construction is accounted for separately.

Cohort	Phase	Interactions
24 BabyAI confirmation policies	training	75497472
24 BabyAI confirmation policies	development	449800
24 BabyAI confirmation policies	test	807386
24 WaterWorld confirmation policies	training	72000000
WaterWorld campaign	total charged including evaluations/checks	82260735
WaterWorld campaign	charged beyond training	10260735

E.4 Learning-curve reconstruction and aggregation

The learning-curve figures are reconstructed from completed runs; plotting uses no additional policy training, interface fitting, or evaluation rollouts. BabyAI contains 24 policy fits: four arms, three interface/grounder bundles, and two policy seeds per arm and bundle. The WaterWorld learning curves contain 20 completed policies: four arms and five policy seeds per arm. The x-axis counts policy-training environment interactions, not offline supervision or total pipeline computation; those costs are reported separately in the preceding tables.

Recorded measurements. BabyAI records 1,536 updates at intervals of 2,048 interactions, ending at 3,145,728. At an update with C completed episodes, the return and completion summaries use episodes $\max(0, C-100), \ldots, C-1$ in callback order. The reconstructed completion summary is checked against the saved update log. The main-text values at approximately one million interactions refer to the recorded point at 999,424, without interpolation.

The main WaterWorld learning figure uses each recorded training rollout: 250 points per policy, with 8,000 interactions per point. In each trailing ten-rollout window, the common normalized native reward is $(S - T/10^5)/(S + F + 8)$, where S is the number of native successes, F is the number of time-limit failures, and T is the number of interactions. The denominator counts every episode intersecting the window, including the eight still running at its end. The policy curves average five seeds with sample-SD bands. Every interaction contributes to the numerator even when its episode has not ended. The first recorded point, at 8,000 interactions, is -0.01 for every policy. There is no zero-step training measurement; a separate paired DEV evaluation checks the untrained policy. The additional main-text WaterWorld completion panel uses the same trailing ten-rollout training windows. In each window, all native success and time-limit events are counted. There are always eight ongoing native episodes at the window's end, one per parallel environment. The displayed observed completion fraction is $S/(S + F + 8)$, with S successes and F time-limit failures in the window. These eight ongoing episodes remain in the denominator as not yet successful, so early fast successes cannot alone yield a 100% rate. This window-observed fraction is distinct from the eventual success probability of completed episodes and from the paired DEV evaluation.

For audit purposes, WaterWorld also retains historical 50,000-interaction episode-start bins. With eight parallel environments, the start index is the recorded global end step minus eight times the

episode length. These bins are retrospective: outcomes become known only when episodes end. They do not drive the active reward or completion figures. Administrative cuts are excluded. For an episode of length L, the common native objective in units of 10^5 is $\mathbf{1}\{\text{success}\} - L/10^5$. The recorded method-specific policy reward can include guidance shaping and is not used for cross-method return plots. This fixed scaling is common to all four methods. The additional dense WaterWorld view shows the same window-observed normalized reward using one rollout per point.

Aggregation and missingness. For BabyAI, let $y_{b,p}(t)$ be the trailing episode summary for policy seed $p \in \{1, 2\}$ in fitted bundle $b \in \{1, 2, 3\}$. We first form $\bar{y}_b(t) = \frac{1}{2}\sum_p y_{b,p}(t)$, then plot

$$\mu(t) = \frac{1}{3}\sum_{b=1}^{3} \bar{y}_b(t), \qquad s(t) = \sqrt{\frac{1}{2}\sum_{b=1}^{3}\big(\bar{y}_b(t) - \mu(t)\big)^2}.$$

Bands are $\mu(t) \pm s(t)$, not confidence intervals or significance tests. All six BabyAI policy values must exist at a displayed aggregate point. WaterWorld instead uses the arithmetic mean of five policy seeds. Its reward and completion rollouts have complete coverage at each recorded point. Bands show sample SD across seeds; individual traces are omitted. Undefined measurements remain missing; no zero filling, extrapolation, or outcome-based exclusion is applied. Completion-rate bands are visually clipped at zero and one; the underlying means and SDs are unchanged.

Separate evaluation panels. BabyAI has six saved greedy-action development checkpoints, each using eight episodes per policy. WaterWorld has eleven paired development checkpoints from zero to two million interactions, each with twelve episodes per policy under sampled and deterministic actions separately. Their unsmoothed native return and completion curves are shown separately from training histories. The earlier WaterWorld terminal TEST panels come from another confirmation cohort and are not pooled with the development results. Native Dyck has one fitted interface and two policy seeds; its fixed-count development curves use only saved observations and descriptive across-policy SD, with warmup and curriculum boundaries marked.

Reproducible figure data. The BabyAI curve asset directory contains per-policy, within-bundle, and aggregate CSVs together with metric definitions and source hashes. The active WaterWorld inputs comprise the original rollout, episode, start-bin, evaluation, and summary files; the 240 underlying source files are hash-checked. Earlier WaterWorld curve exports are retained as historical inputs, and original terminal-result tables retain their earlier-cohort denominators.

F Additional BabyAI results

F.1 Policy and mission-level outcomes

Figure F.2 summarizes policy rates across the fitted bundles on the held-out mission panel.

F.2 Training and development learning curves

Figure 3(c) shows BabyAI native training completion alongside the reward curves in the main text. At the final training window, LO-guided, context-only, and the oracle reference complete 99.33%, 99.00%, and 100.00% of episodes, compared with 89.17% for task-conditioned PPO. Thus, the reward separation is accompanied by a difference in task completion, not merely a difference in method-specific guidance rewards.

Figure F.3 compares all methods at the six saved development checkpoints, each using eight episodes per policy. At 1,048,576 interactions, native development return is 0.657 ± 0.048 for LO-guided, 0.453 ± 0.133 for context-only, and 0.110 ± 0.061 for task-conditioned PPO; the oracle reference gives 0.655 ± 0.079. This separate evaluation supports an early learning advantage for the learned interface, but not a fixed ordering of the guided and context-only arms at every checkpoint: at 2,097,152 interactions, their means are 0.679 and 0.703, respectively.

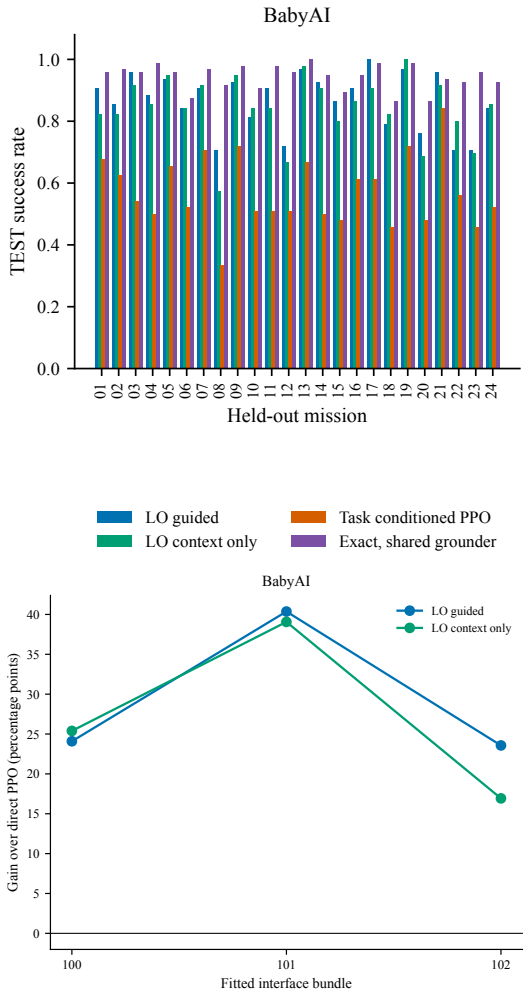


Figure F.1: Frozen-interface transfer in BabyAI. (A) Completion across the 24 held-out missions, in the fixed evaluation order. Each cell averages two policies within each bundle and then three bundle means. (B) Paired completion differences relative to task-conditioned PPO for all three bundles.

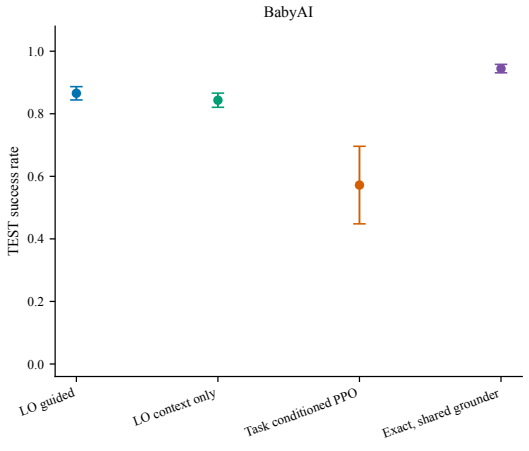


Figure F.2: BabyAI held-out-mission completion. Markers show the mean of six policy rates per arm and error bars their descriptive sample SD. Three fitted bundles each contribute two policies. The direct control receives no offline interface supervision.

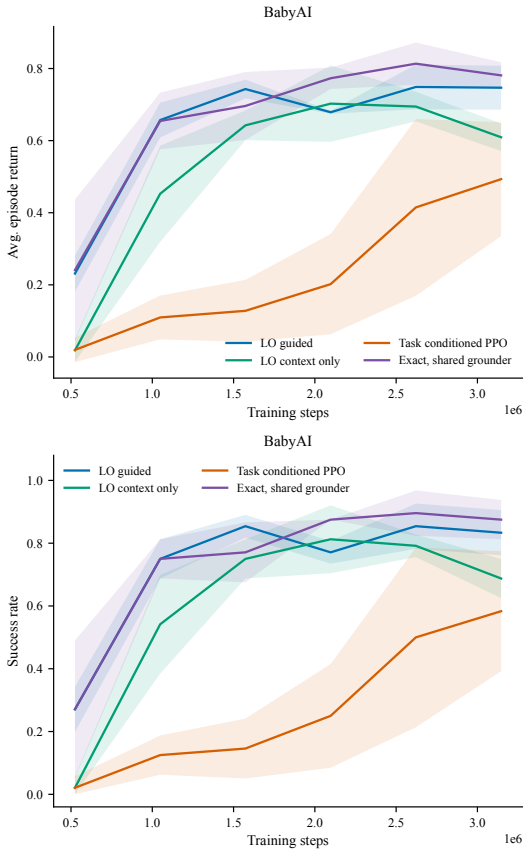


Figure F.3: Saved BabyAI development evaluations. Top: native episode return. Bottom: task completion. Each of the six checkpoints uses eight episodes per policy with greedy actions. Lines connect measured checkpoints without smoothing; bands show sample SD across three bundle means, each averaging two policies. These panels are separate from training and terminal TEST.

G ADDITIONAL WATERWORLD RESULTS

G.1 EARLIER CONFIRMATION-COHORT TERMINAL TESTS

Figures G.1 and G.2 show every policy and bundle mean under deterministic and sampled action selection. Tables 18 and 19 give the corresponding numerators and denominators. Each policy receives 64 deterministic and 16 sampled evaluation layouts. Across the six policy results per arm, LO-guided completes 382/384 deterministic and 93/96 sampled episodes; context-only completes 380/384 and 92/96, compared with 0/384 and 3/96 for recurrent PPO. Exact semantics with shared learned grounding gives 382/384 and 92/96.

These earlier three-bundle results use the corrected collection-and-recall implementation and newly fitted domain-specific interfaces. They are separate from the five-seed learning curves below. The task contains eight collection and eight recall events. Although formal training and development use a 50/20 split of parity words, PPO can encounter all 70 balanced orders. This split therefore does not establish whole-agent unseen-order transfer. The recurrent control has a different policy architecture from the interface arms and no offline interface supervision.

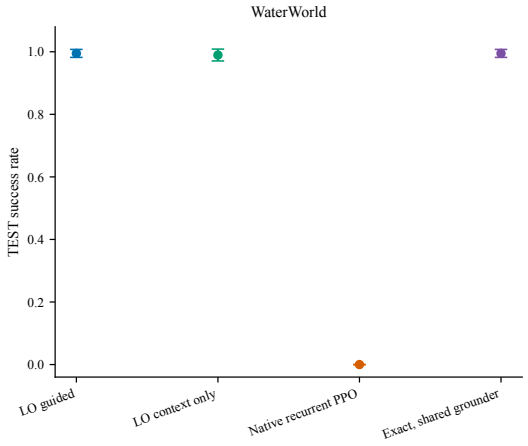


Figure G.1: WaterWorld terminal completion with deterministic actions. Markers show the mean of six policy rates per arm; error bars show their descriptive sample SD. Each policy is evaluated on 64 layouts at fixed task size.

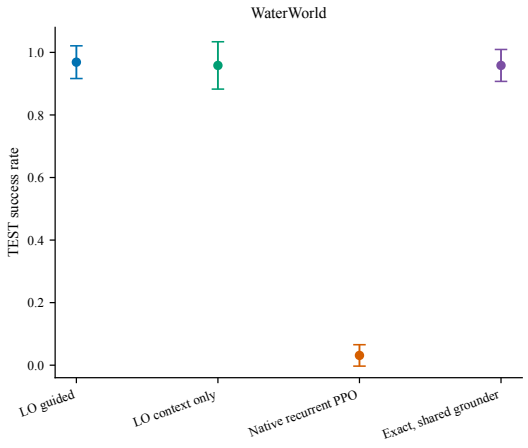


Figure G.2: WaterWorld terminal completion with sampled actions. Each policy is evaluated on 16 layouts. Markers show the mean of six policy rates per arm and error bars their descriptive sample SD; these results are not pooled with deterministic evaluation.

Table 18: WaterWorld deterministic outcomes for every fitted bundle, policy seed, and arm. Each policy has 64 evaluation episodes.

Method	Bundle	Policy	Successes / episodes	Completion (%)
LO-guided	0	200	64/64	100.00
Exact + shared learned grounder	0	200	64/64	100.00
LO context-only	0	200	64/64	100.00
Recurrent no-LO PPO	0	200	0/64	0.00
LO-guided	0	201	64/64	100.00
Exact + shared learned grounder	0	201	64/64	100.00
LO context-only	0	201	64/64	100.00
Recurrent no-LO PPO	0	201	0/64	0.00
LO-guided	1	202	64/64	100.00
Exact + shared learned grounder	1	202	64/64	100.00
LO context-only	1	202	61/64	95.31
Recurrent no-LO PPO	1	202	0/64	0.00
LO-guided	1	203	62/64	96.88
Exact + shared learned grounder	1	203	62/64	96.88
LO context-only	1	203	64/64	100.00
Recurrent no-LO PPO	1	203	0/64	0.00
LO-guided	2	204	64/64	100.00
Exact + shared learned grounder	2	204	64/64	100.00
LO context-only	2	204	63/64	98.44
Recurrent no-LO PPO	2	204	0/64	0.00
LO-guided	2	205	64/64	100.00
Exact + shared learned grounder	2	205	64/64	100.00
LO context-only	2	205	64/64	100.00
Recurrent no-LO PPO	2	205	0/64	0.00

Table 19: WaterWorld sampled-action outcomes for every fitted bundle, policy seed, and arm. Each policy has 16 evaluation episodes.

Method	Bundle	Policy	Successes / episodes	Completion (%)
LO-guided	0	200	16/16	100.00
Exact + shared learned grounder	0	200	16/16	100.00
LO context-only	0	200	16/16	100.00
Recurrent no-LO PPO	0	200	1/16	6.25
LO-guided	0	201	16/16	100.00
Exact + shared learned grounder	0	201	16/16	100.00
LO context-only	0	201	16/16	100.00
Recurrent no-LO PPO	0	201	0/16	0.00
LO-guided	1	202	15/16	93.75
Exact + shared learned grounder	1	202	15/16	93.75
LO context-only	1	202	15/16	93.75
Recurrent no-LO PPO	1	202	0/16	0.00
LO-guided	1	203	16/16	100.00
Exact + shared learned grounder	1	203	16/16	100.00
LO context-only	1	203	16/16	100.00
Recurrent no-LO PPO	1	203	0/16	0.00
LO-guided	2	204	16/16	100.00
Exact + shared learned grounder	2	204	15/16	93.75
LO context-only	2	204	16/16	100.00
Recurrent no-LO PPO	2	204	1/16	6.25
LO-guided	2	205	14/16	87.50
Exact + shared learned grounder	2	205	14/16	87.50
LO context-only	2	205	13/16	81.25
Recurrent no-LO PPO	2	205	1/16	6.25

G.2 TRAINING AND DEVELOPMENT LEARNING CURVES

The completed WaterWorld cohort contains five policy seeds per method and two million training interactions per seed. Figure 3(d) shows observed native completion in trailing ten-rollout windows. Its denominator includes all native episodes that intersect a window, including the eight episodes still running at its end. Thus the first recorded point is zero in every arm, without selecting only the first short successful episodes. At two million interactions, equal-seed means are 95.7% for LO-guided, 95.5% for the exact reference, 71.3% for context-only, and 17.7% for recurrent PPO. These are window-observed training completion fractions, not held-out episode success probabilities.

Figure G.3 shows eleven paired development checkpoints from zero through two million interactions, evaluated separately under sampled and deterministic actions. Each seed has twelve fixed episodes per checkpoint and action mode. At the final checkpoint, LO-guided and the exact reference complete all 60 episodes under both modes; context-only completes 49/60, while recurrent PPO completes 26/60 sampled and 20/60 deterministic episodes. These are development measurements and are not pooled with the earlier confirmation-cohort terminal tests above. The plotted return is the common native task objective, excluding each method's shaped policy reward.

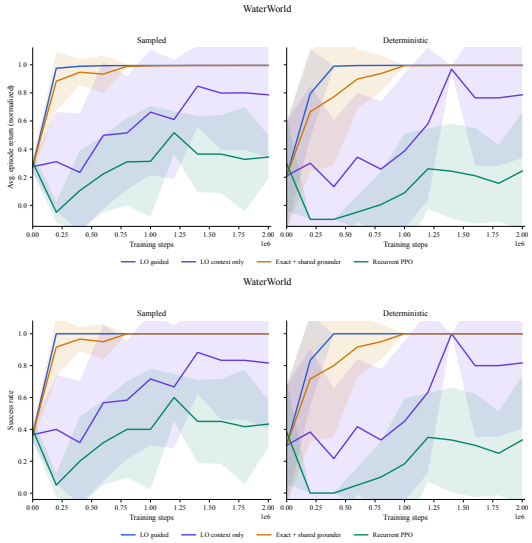


Figure G.3: WaterWorld paired development evaluations. Top: common native task-objective return in units of 10^5; bottom: native completion. Each panel separates sampled and deterministic action selection at eleven saved checkpoints, with twelve episodes per policy seed and mode. Lines average the five seeds, with sample-SD bands. No terminal TEST episodes are inserted.

H Additional parameter-generalization results

H.1 Full traces, parameter edits, and per-count queries

Tables 20 and 21 disaggregate the main semantic results by fit. All five LO fits succeed on every evaluated full trace and parameter edit in their respective domains. The learned task machine receives semantic supervision but not LO's additional local-executor targets, and its capacity differs. Exact recompiled machines and exact logic with learned grounding are oracle references, not fitted learned-machine results.

Tables 22 and 23 report query correctness separately for counts 13–16. Their numerators sum correct queries, not complete traces; they provide a finer view of the same semantic study rather than additional fitted models. Parameter edits change the public argument while retaining the history and model weights, and repeated events are preserved during replay.

Table 20: Counter semantic execution by fit. Full-trace and parameter-edit counts are evaluated with frozen models trained on counts 1–12 and tested at 13–16. Exact references and the learned machine retain their distinct supervision and execution contracts.

Method	Fit	Full traces	Parameter edits
Parameterized LO	0	56/56	6/6
Learned task machine	0	8/56	0/6
Exact known-structure recompiled	0	56/56	6/6
Exact logic + learned grounder	0	56/56	6/6
Parameterized LO	1	56/56	6/6
Learned task machine	1	6/56	0/6
Exact known-structure recompiled	1	56/56	6/6
Exact logic + learned grounder	1	56/56	6/6
Parameterized LO	2	56/56	6/6
Learned task machine	2	0/56	0/6
Exact known-structure recompiled	2	56/56	6/6
Exact logic + learned grounder	2	56/56	6/6
Parameterized LO	3	56/56	6/6
Learned task machine	3	8/56	0/6
Exact known-structure recompiled	3	56/56	6/6
Exact logic + learned grounder	3	56/56	6/6
Parameterized LO	4	56/56	6/6
Learned task machine	4	8/56	0/6
Exact known-structure recompiled	4	56/56	6/6
Exact logic + learned grounder	4	56/56	6/6

Table 21: Dyck semantic execution by fit. Full-trace and parameter-edit counts are evaluated with frozen models trained on counts 1–12 and tested at 13–16. Exact references and the learned machine retain their distinct supervision and execution contracts.

Method	Fit	Full traces	Parameter edits
Parameterized LO	0	96/96	9/9
Learned task machine	0	4/96	0/9
Exact known-structure recompiled	0	96/96	9/9
Exact logic + learned grounder	0	96/96	9/9
Parameterized LO	1	96/96	9/9
Learned task machine	1	4/96	0/9
Exact known-structure recompiled	1	96/96	9/9
Exact logic + learned grounder	1	96/96	9/9
Parameterized LO	2	96/96	9/9
Learned task machine	2	0/96	0/9
Exact known-structure recompiled	2	96/96	9/9
Exact logic + learned grounder	2	96/96	9/9
Parameterized LO	3	96/96	9/9
Learned task machine	3	4/96	0/9
Exact known-structure recompiled	3	96/96	9/9
Exact logic + learned grounder	3	96/96	9/9
Parameterized LO	4	96/96	9/9
Learned task machine	4	4/96	0/9
Exact known-structure recompiled	4	96/96	9/9
Exact logic + learned grounder	4	96/96	9/9

Table 22: Counter query-level correctness at each held-out count. Counts pool the five formal fits. This metric counts individual correct queries and is distinct from full-trace correctness.

Method	n	Correct queries / queries	Query accuracy (%)
Parameterized LO	13	1670/1670	100.00
Parameterized LO	14	1780/1780	100.00
Parameterized LO	15	1890/1890	100.00
Parameterized LO	16	2000/2000	100.00
Learned task machine	13	1131/1670	67.72
Learned task machine	14	1214/1780	68.20
Learned task machine	15	1266/1890	66.98
Learned task machine	16	662/2000	33.10
Exact known-structure recompiled	13	1670/1670	100.00
Exact known-structure recompiled	14	1780/1780	100.00
Exact known-structure recompiled	15	1890/1890	100.00
Exact known-structure recompiled	16	2000/2000	100.00
Exact logic + learned grounder	13	1670/1670	100.00
Exact logic + learned grounder	14	1780/1780	100.00
Exact logic + learned grounder	15	1890/1890	100.00
Exact logic + learned grounder	16	2000/2000	100.00

Table 23: Dyck query-level correctness at each held-out count. Counts pool the five formal fits. This metric counts individual correct queries and is distinct from full-trace correctness.

Method	n	Correct queries / queries	Query accuracy (%)
Parameterized LO	13	2085/2085	100.00
Parameterized LO	14	2220/2220	100.00
Parameterized LO	15	2355/2355	100.00
Parameterized LO	16	2490/2490	100.00
Learned task machine	13	528/2085	25.32
Learned task machine	14	553/2220	24.91
Learned task machine	15	573/2355	24.33
Learned task machine	16	599/2490	24.06
Exact known-structure recompiled	13	2085/2085	100.00
Exact known-structure recompiled	14	2220/2220	100.00
Exact known-structure recompiled	15	2355/2355	100.00
Exact known-structure recompiled	16	2490/2490	100.00
Exact logic + learned grounder	13	2085/2085	100.00
Exact logic + learned grounder	14	2220/2220	100.00
Exact logic + learned grounder	15	2355/2355	100.00
Exact logic + learned grounder	16	2490/2490	100.00

H.2 NATIVE DYCK COMPLETION AND DEPTH COVERAGE

The native study uses one formal/grounder fit and two policies per arm. The interface is trained through count 12, whereas policy training uses counts no larger than 4. Read-only probes at counts 1–8 and 9–12 precede the final 13–16 panel. The numerical range had been examined in earlier research; the final layouts are fresh and the evaluated policies remain fixed.

Figure H.1 reports task completion and the fraction of episodes whose maximum attained depth equals the requested count. Table 24 also reports their intersections. At $n = 16$, completion is 63/64 and 62/64 for the two LO policies, while both reach depth 16 in 62/64 episodes. Completing a task does not by itself establish that the requested depth was reached.

For each LO policy, 253 of the 256 final histories contain one accepted color, two contain two colors, and one contains no accepted color, including failures. The native result is therefore predominantly single-color; mixed-color semantic correctness is established in the separate offline study. The actor's interface context contains target geometry, with its semantic channels zeroed. Learned semantics affect target selection and local-progress/ event-bonus rewards, whose complete sum is not purely potential-based.

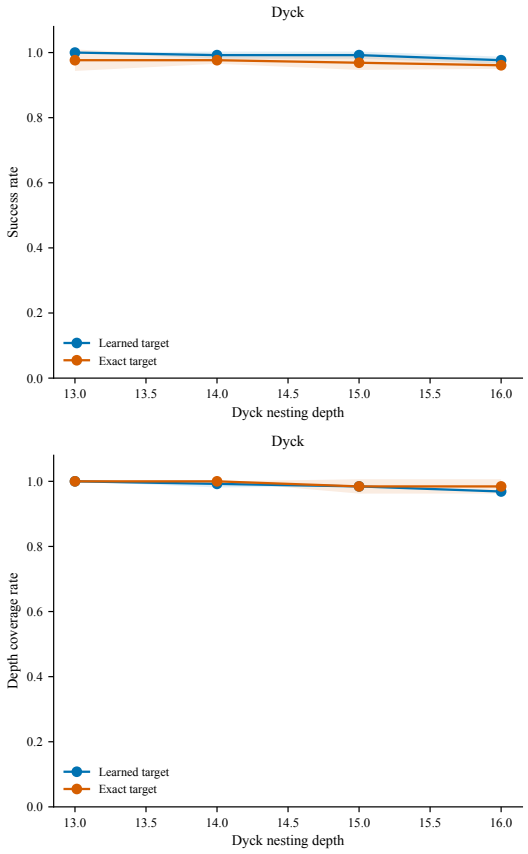


Figure H.1: Frozen native Dyck evaluation. Top: task completion. Bottom: the fraction of episodes attaining maximum depth exactly equal to the requested count. Each point uses 64 episodes. Policy training ends at count 4 and formal-interface training at count 12. Both policy seeds share one formal/grounder fit. The exact-target arm has privileged target access.

Table 24: Native Dyck completion and depth coverage by requested count and policy seed. The table distinguishes completion, reaching depth exactly n, and their intersections. The exact-target diagnostic uses privileged target information.

Method	Policy	n	Episodes	Completed	Reached depth n
LO	0	13	64	64	64
LO	0	14	64	64	63
LO	0	15	64	64	63
LO	0	16	64	63	62
LO	1	13	64	64	64
LO	1	14	64	63	64
LO	1	15	64	63	63
LO	1	16	64	62	62
Exact target	0	13	64	61	64
Exact target	0	14	64	63	64
Exact target	0	15	64	61	62
Exact target	0	16	64	62	62
Exact target	1	13	64	64	64
Exact target	1	14	64	62	64
Exact target	1	15	64	63	64
Exact target	1	16	64	61	64

Method	Policy	n	Completed and reached n	Completed below n	Failed after reaching n
LO	0	13	64	0	0
LO	0	14	63	1	0
LO	0	15	63	1	0
LO	0	16	62	1	0
LO	1	13	64	0	0
LO	1	14	63	0	1
LO	1	15	62	1	1
LO	1	16	61	1	1
Exact target	0	13	61	0	3
Exact target	0	14	63	0	1
Exact target	0	15	61	0	1
Exact target	0	16	62	0	0
Exact target	1	13	64	0	0
Exact target	1	14	62	0	2
Exact target	1	15	63	0	1
Exact target	1	16	61	0	3

H.3 FIXED-COUNT DEVELOPMENT LEARNING CURVES

Figures H.2 and H.3 report saved native development returns for counts 1–4 from the verified training parents of the final policies. LO reaches a mean return of 1.000 at count 1 by 524,288 interactions and at counts 3 and 4 by 1,310,720 interactions, showing that useful control is learned with the frozen interface as the curriculum expands. The privileged exact-target diagnostic also reaches high returns, and the curves do not establish a consistent learning-speed advantage for LO. Returns are not monotone: at the final saved checkpoint, LO's means at counts 2, 3, and 4 are 0.863, 0.938, and 0.938, respectively. These within-curriculum curves complement, rather than replace, the frozen count-13–16 evaluations above.

The four count panels are kept separate because changing the curriculum changes the task distribution. Warmup ends at 65,536 interactions; the maximum training count rises to 2 at 524,288 and to 4 at 1,048,576. The last saved development checkpoint is 1,441,792, before the 1,572,864-step training budget; curves are not extended to that budget. Both policy seeds share one formal/grounder fit. Bands therefore describe sample SD across two policies only, not variability across independently fitted operators. Contact diagnostics are not used as scalar return records.

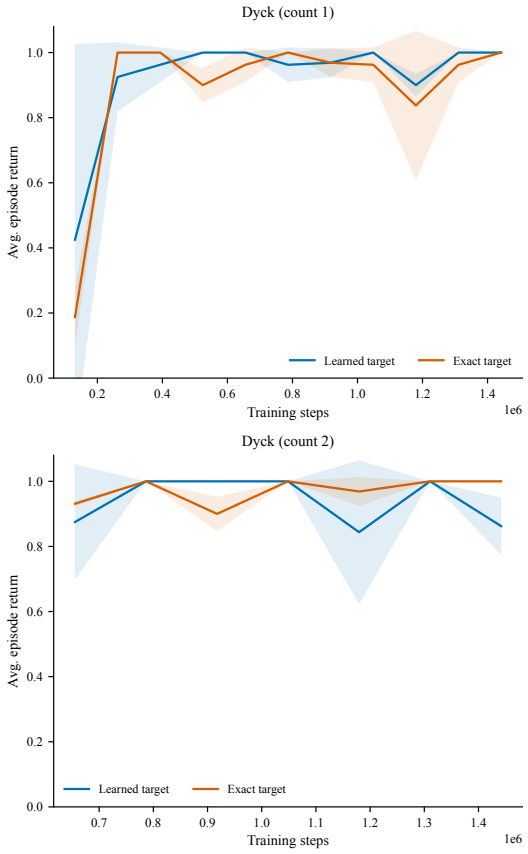


Figure H.2: Dyck native development return at fixed counts 1 (top) and 2 (bottom). Lines and bands show the mean and descriptive sample SD of two policy seeds. One frozen formal/grounder fit is shared. The exact-target condition has privileged target access, unlike the shared-learned-grounder reference in the external domains. Only saved greedy-action checks are shown.

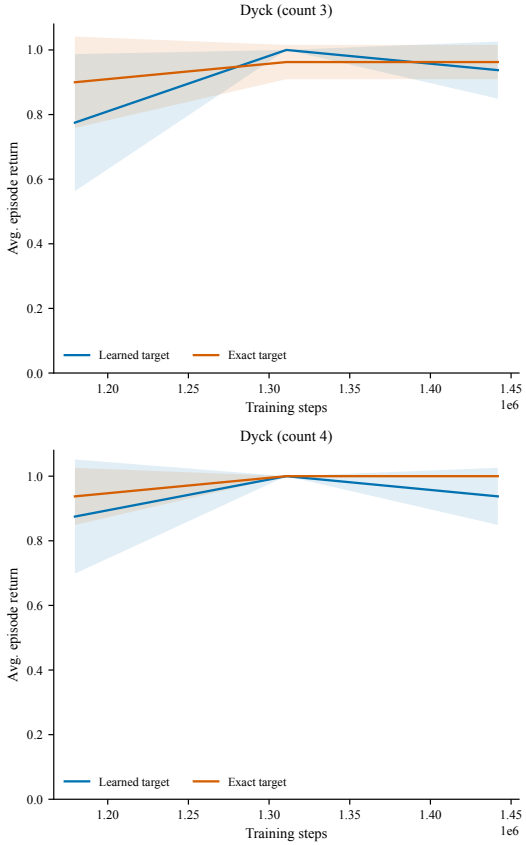


Figure H.3: Dyck native development return at fixed counts 3 (top) and 4 (bottom). Only three saved checkpoints per policy are available after these counts enter the curriculum. Means and descriptive two-policy SD use the same convention as Figure H.2. The comparison is with a privileged exact-target diagnostic and is separate from larger-count terminal evaluations.

I DISCUSSION AND LIMITATIONS

The experiments support learning a reusable map from specifications to grounded reward-interface functions. Their scope differs by setting: BabyAI measures held-out mission compositions, Counter/Dyck measure larger numerical arguments, and WaterWorld provides a separate fixed-size continuous-control application. Each domain uses its own fitted operator; these experiments do not establish frozen cross-domain weight transfer. Finite numerical tests do not establish unbounded recognition, and the native Dyck result uses one formal fit with predominantly single-color histories.

Offline monitor supervision is an important assumption. The numerical executor also uses exact generic memory and arithmetic operations, while the direct PPO controls do not receive the interfaces' offline labels. Consequently, these comparisons establish utility under the stated training conditions rather than an architectural advantage under matched supervision. An equally supervised shared-versus-per-task comparison is not part of the reported evidence. Exact semantic references identify remaining room for improving the learned interfaces, particularly in BabyAI, where high native completion coexists with imperfect full-trace semantic predictions.

The control results depend on the interface-to-policy and reward construction. Context-only removes added guidance reward but retains learned memory and supervision. Native Dyck includes a non-potential event bonus, so the complete reward is not covered by policy-invariance guarantees for potential shaping alone. Earlier failed configurations further show that accurate semantic execution is not sufficient for every policy-training procedure. The current experiments do not establish robustness to imperfect monitor labels or comprehensive insensitivity to loss weights.