COMPILING TASK-EVENT CIRCUITS FOR OFFLINE GOAL-CONDITIONED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Goal-conditioned policies often reach nearby goals yet fail when a task requires composing many distinct changes. We present the Algebraic Task Quotient (ATQ), a compiler that extracts repeated task changes from offline replay. ATQ represents task progress with persistent variables, which we call registers. It learns how recurring task-level events update these registers and compiles the learned rules into a task-event circuit. Each path through the circuit represents a possible sequence of task changes. At test time, a planner selects an event sequence, a policy executes one local request, and feedback updates the circuit state.

On all eight OGBench Puzzle tasks, ATQ reaches 95.3–100.0% success with eight independently trained GCIQL policies. Under the same policies and test conditions, Test-Time Graph Search reaches 0.2–44.1% on the 4x4–4x6 layouts, whereas ATQ reaches 95.3–100.0%. ATQ also exceeds the cited published scores on all 4x5 and 4x6 tasks. On the larger puzzles, 98.4–100.0% of its requests apply an event in a register configuration where that event was absent from compiler training data. Episodes composed entirely of these new combinations still succeed 95.3–100.0% of the time. Beyond Puzzle, ATQ raises success in navigation, object manipulation, and soccer, with gains of up to 27.2 percentage points. These results show that replay can reveal event rules that support composition beyond search over recorded states.

1 INTRODUCTION

A goal-conditioned policy may possess every local skill needed for a task and still fail to finish it. The policy can execute a local goal, but it does not know which task change should happen next.

Offline replay can reveal the missing planning structure. Different trajectories may contain the same kind of task change in different configurations. These repeated local changes can reveal a rule that a planner can reuse and compose with other rules. No single trajectory needs to solve the new target.

In this paper, we introduce the Algebraic Task Quotient (ATQ), a compiler that learns reusable task-event rules from offline replay and represents them as a circuit for goal-directed planning. We treat replay as a collection of program traces. We borrow the terms *register* and *circuit* from sequential digital circuits. Registers store task progress, and event-conditioned update rules determine the next register state. ATQ searches a supplied set of candidate task variables and event models. Replay selects the variables that persist across time, the events that recur, and the rule associated with each event. The compiler emits a task-event circuit. The circuit records the current task configuration and predicts how each event changes it. A planner searches this circuit, and a linker converts the next event into a local goal for the policy. The planner observes the realized change and replans after every attempt.

The circuit separates two questions. Its operators describe what an event does to the task. We call the existing goal-conditioned policy the *executor*. It determines how to realize that event in the original action space. This separation differs from replay-graph planning, which searches among recorded states. ATQ can apply a learned event rule in a task configuration that replay never paired with that event.

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.

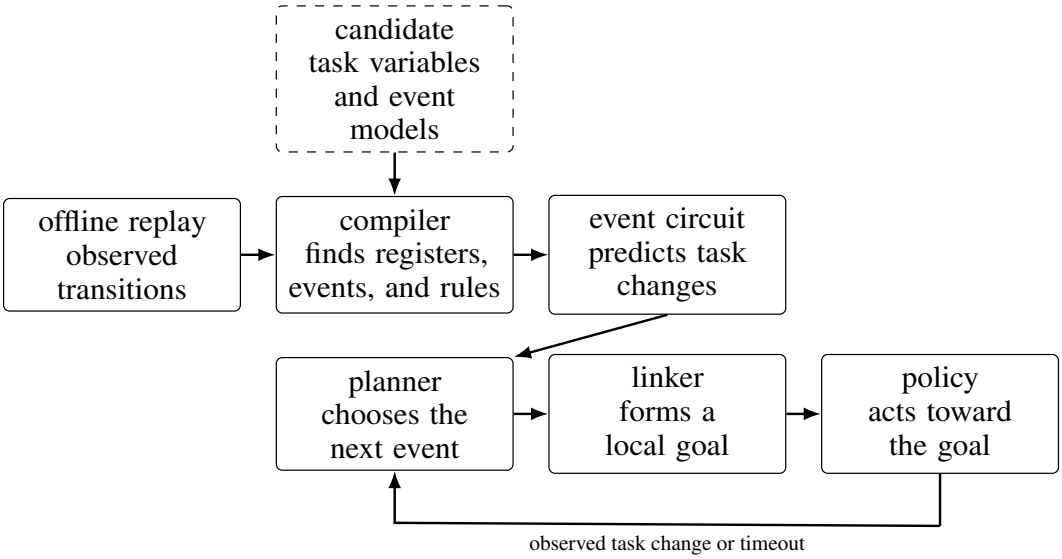

Figure 1: The user supplies candidate variable and operator types. Replay selects their concrete instances. At test time, the circuit plans task changes and the policy realizes one local goal at a time. Feedback updates the circuit after every attempt.

We formalize when this separation is valid. An exact task representation must preserve success, event applicability, and the task state produced by each event. These local conditions preserve every event sequence. They also define a canonical quotient. Two task configurations are equivalent when the same future event sequences can solve both.

We evaluate ATQ on OGBench, a benchmark for offline goal-conditioned reinforcement learning (Park et al., 2025). The experiments span navigation, manipulation, soccer, and spatial assignment. Puzzle, a robotic button-pressing family within OGBench, provides the main test of composition. Each button produces a reusable task change, but reaching a target pattern may require a plan absent from any single replay trajectory. We evaluate all eight Puzzle tasks using the same policies and compare ATQ directly with GCIQL and Test-Time Graph Search (TTGS) (Opryshko et al., 2026).

Our contributions are threefold.

- We introduce a compiler that learns task registers and reusable event rules from offline replay. It sends each planned event to the policy as a local goal and replans from the observed outcome.
- We give conditions under which the circuit preserves goal reachability. We also show when replay determines an operator on event combinations that were not observed during compiler training.
- We obtain 95.3–100.0% success on all eight Puzzle tasks and outperform TTGS under the same test conditions on every task beyond the smallest layout. Experiments in other OGBench families test how the same idea transfers beyond XOR.

2 The ATQ Method

2.1 Event-level goal reaching

Consider a Puzzle board whose lights form a binary vector. Pressing one switch may change several lights at once. We call the light vector a *register*, the switch press an *event*, and the resulting binary update an *operator*. The same language applies beyond Puzzle. A register records the part of the task that persists, and an event denotes a repeated change that the executor can attempt. An event may require many motor actions rather than one environment action.

Let o_t be the observation at motor-control time t. The compiler partitions each replay trajectory at candidate event boundaries. At boundary k, an extractor maps the available observation and action history to a task configuration

$$u_k = \widehat{\phi}(H_k^o, g), \tag{1}$$

where H_k^o contains the observations and actions available at that boundary. In domains with persistent discrete variables, we write $u_k = (r_k, g)$ for the register vector r_k and goal register g.

Let $\mathcal{X}$ contain histories paired with their goals. Let $a : \mathcal{X} \to \{0, 1\}$ indicate task success. An event $e \in \mathcal{E}$ has a partial deterministic transition $T_e : D_e^{\mathcal{X}} \to \mathcal{X}$. The selected extractor is exact for this event interface when, for every x, x' with $\phi(x) = \phi(x')$ and every e,

$$\begin{aligned} a(x) &= a(x'), \\ \mathbf{1}[x \in D_e^{\mathcal{X}}] &= \mathbf{1}[x' \in D_e^{\mathcal{X}}], \\ \phi(T_e x) &= \phi(T_e x') \quad \text{whenever } x, x' \in D_e^{\mathcal{X}}. \end{aligned} \tag{2}$$

The three lines require states merged by ϕ to agree on success, event applicability, and the task configuration after the event.

Theorem 1 (Exact compiler contract)**.** *There exists an exact deterministic task-event circuit on $\phi(\mathcal{X})$ factoring the source transitions through ϕ if and only if eq. (2) holds. In that case there are well-defined functions τ, G_e, and F_e such that*

$$\tau(\phi(x)) = a(x), \qquad G_e(\phi(x)) = \mathbf{1}[x \in D_e^{\mathcal{X}}], \qquad F_e(\phi(x)) = \phi(T_e x) \quad (x \in D_e^{\mathcal{X}}). \tag{3}$$

The one-step equality extends by induction to every executable event word. Thus the circuit preserves acceptance after every such word. The kernel of ϕ refines source continuation equivalence.

The proof is in appendix A. The theorem reduces exactness to three local conditions that can be checked on transitions. If any condition fails, the extractor cannot represent the specified event interface exactly.

Each learned event has a guard $G_e(u)$ with domain $D_e = \{u \in \mathcal{U} : G_e(u) = 1\}$ and an operator $F_e : D_e \to \mathcal{U}$. When $u = (r, g)$, the operator does not change the goal.

$$F_e(r, g) = \big(f_e(r, g), g\big), \qquad (r, g) \in D_e. \tag{4}$$

We totalize each partial operator by sending unsupported applications to an artificial rejecting state.

$$\overline{\mathcal{U}} = \mathcal{U} \cup \{\bot\}, \qquad \overline{\tau}(u) = \begin{cases} \tau(u), & u \in \mathcal{U}, \\ 0, & u = \bot, \end{cases}$$

$$\overline{F}_e(u) = \begin{cases} u, & u \in \mathcal{U} \text{ and } \tau(u) = 1, \\ F_e(u), & u \in \mathcal{U},\ \tau(u) = 0, \text{ and } G_e(u) = 1, \\ \bot, & \text{otherwise}, \end{cases} \tag{5}$$

The rejecting state represents an invalid event request. Acceptance is absorbing, so execution stops at the first successful prefix.

For an event word $w = e_1 \cdots e_K$, let $\overline{F}_w = \overline{F}_{e_K} \circ \cdots \circ \overline{F}_{e_1}$ and let $\overline{F}_\epsilon$ be the identity map for the empty word ϵ. The successful continuation language of u is

$$\mathcal{L}_u = \{w \in \mathcal{E}^* : \overline{\tau}(\overline{F}_w(u)) = 1\}. \tag{6}$$

Two configurations are continuation equivalent when their languages are equal. The ideal task-event quotient is $\mathcal{Q}^* = \overline{\mathcal{U}}/\equiv$, where $u \equiv v$ if and only if $\mathcal{L}_u = \mathcal{L}_v$. Its transition and success predicate are

$$\delta^*([u], e) = [\overline{F}_e(u)], \qquad \tau^*([u]) = \overline{\tau}(u). \tag{7}$$

Theorem 2 (Canonical continuation quotient)**.** *Continuation equivalence is stable under every totalized event operator. The quotient preserves current success, whether each event word reaches acceptance, the existence of a successful continuation, and the minimum unit-cost word length to acceptance. For any equivalence $\approx$ that preserves success after every event word, $u \approx v$ implies $u \equiv v$.*

This theorem follows the residual-language construction (Hopcroft et al., 2006). It gives the coarsest representation preserving successful event words. Two configurations may be merged precisely when every future event sequence has the same success outcome.

2.2 THE OFFLINE COMPILER

For each environment family d, we specify a finite candidate library and the request type accepted by its executor.

$$\mathfrak{L}_d = (\Phi_d, \Psi_d, \mathcal{H}_d, \mathcal{Y}_d). \tag{8}$$

Here Φ_d contains register extractors, Ψ_d contains event identifiers, $\mathcal{H}_d$ contains guarded operators, and $\mathcal{Y}_d$ is the executor's request space. The library determines what kinds of regularity the compiler can express. Replay selects concrete registers, events, operator parameters, and request prototypes. With a separate development split, selection proceeds in four steps.

$$\begin{aligned} \widehat{\phi} &= \mathrm{SelectRegister}(\mathcal{D}_{\mathrm{fit}}; \Phi), \\ \widehat{\psi} &= \mathrm{SelectEvent}(\mathcal{D}_{\mathrm{fit}}; \Psi, \widehat{\phi}), \\ \mathcal{H}_{\mathrm{fit}} &= \mathrm{FitOperators}(\mathcal{D}_{\mathrm{fit}}; \widehat{\phi}, \widehat{\psi}, \mathcal{H}), \\ \widehat{h} &= \mathrm{Select}(\mathcal{D}_{\mathrm{dev}}; \mathcal{H}_{\mathrm{fit}}). \end{aligned} \tag{9}$$

Subscripts d are omitted in eq. (9). The test split is used only to measure the selected compiler. Exact split sizes and candidate libraries appear in the appendix.

The selectors favor the smallest model that predicts held-out transitions. One form keeps all candidates within an error band and chooses the smallest.

$$\mathcal{H}_\epsilon = \left\{ h : \widehat{R}_{\mathrm{dev}}(h) \le \min_{h'} \widehat{R}_{\mathrm{dev}}(h') + \epsilon \right\}, \qquad \widehat{h} = \arg\min_{h \in \mathcal{H}_\epsilon} \mathrm{Size}(h). \tag{10}$$

Other families minimize a description-length score $[L_{\mathrm{err}}(h) + L_{\mathrm{model}}(h)]/N_h$. Here N_h is the number of binary output labels encoded for candidate h. The first term encodes observed prediction errors, and the second charges $\log_2(N_h + 1)$ per discrete description unit. For equally accurate candidates evaluated on the same output labels, this favors the smaller operator.

Whenever an event can be recognized before its effect, its identifier uses only pre-effect evidence.

$$e_k = \widehat{\psi}(Z_k^-), \qquad r_{k+1} \notin Z_k^-. \tag{11}$$

The successor register is the supervised target for fitting F_e. In Maze and Soccer, an event is instead defined as a transition between two replay-derived places, so both endpoints identify the event.

The candidate library is the main inductive bias. It specifies possible extractor types, operator families, and request schemas. It does not specify the events present in a dataset or a solution to any task. For Puzzle, for example, XOR is one candidate operator. Replay determines the register size, event vocabulary, XOR masks, and switch targets.

Let $\{\widehat{G}_e, \widehat{F}_e\}_{e \in \mathcal{E}}$ be the guards and operators returned by $\widehat{h}$. The goal interface defines success after the compiler selects the register representation. We denote the induced test by $\widehat{\tau}$. Let $\widehat{\mathcal{A}} = \{u : \widehat{\tau}(u) = 1\}$, and let $\widehat{\mathcal{U}}_\perp = \widehat{\mathcal{U}} \cup \{\perp\}$. We define $\overline{\widehat{F}}_e$ by the same convention as eq. (5). It fixes states in $\widehat{\mathcal{A}}$, applies $\widehat{F}_e$ when $\widehat{G}_e = 1$, and otherwise returns $\perp$. The compiler emits the task-event circuit

$$\widehat{\mathcal{C}} = (\widehat{\mathcal{U}}_\perp, \mathcal{E}, \{\widehat{G}_e, \overline{\widehat{F}}_e\}_{e \in \mathcal{E}}, \widehat{\mathcal{A}}), \qquad \widehat{\delta}(u, e) = \overline{\widehat{F}}_e(u). \tag{12}$$

The circuit predicts task changes, not motor actions. A stateful linker converts the next event into a request for the executor.

$$\widehat{\Gamma}_{d,\theta} : \mathcal{E} \times \mathcal{R}_d \times \mathcal{G}_d \times \mathcal{M}_d \longrightarrow \mathcal{Y}_d \times \mathcal{M}_d. \tag{13}$$

Here $\mathcal{R}_d$ is the runtime context space, $\mathcal{G}_d$ is the public goal space, and $\mathcal{M}_d$ stores retry or deferral state. Replay supplies θ, which contains switch targets, object prototypes, waypoints, or action codes. This separation lets the same task-level planner use different motor-control policies.

2.3 WHEN DOES REPLAY IDENTIFY A USEFUL CIRCUIT?

A useful circuit requires a task representation that satisfies eq. (2) and an operator family that contains the repeated changes in replay. Replay can then support planning in two ways. Direct coverage

Algorithm 1 ATQ compilation and closed-loop execution

Require: Offline episodes $\mathcal{D}$, candidate library $\mathfrak{L}_d$, and executor π_{exec}
Split complete episodes using the family-specific protocol
Select register extractor $\widehat{\phi}$ and event identifier $\widehat{\psi}$
Fit candidate guards and operators on the fit transitions
Select operators by held-out accuracy and model complexity
Emit circuit $\widehat{\mathcal{C}}$ and linker parameters θ

Require: History H_k^o, goal g, and linker memory m_k
while time remains **do**
  $u_k \leftarrow \widehat{\phi}(H_k^o, g)$
  if u_k is accepting **then**
    return success
  end if
  $w_k \leftarrow \mathrm{Plan}(\widehat{\mathcal{C}}, u_k)$
  if w_k is empty **then**
    return failure
  end if
  $(y_k, \widetilde{m}_k) \leftarrow \widehat{\Gamma}_{d,\theta}(\mathrm{First}(w_k), H_k^o, g, m_k)$
  Let π_{exec} pursue y_k until an event or timeout is observed
  Update (H_{k+1}^o, m_{k+1}) from the outcome and $\widetilde{m}_k$
  Replan from the observed task configuration
end while
return failure

supports interpolation over observed state–event pairs. A restricted operator family can also support new combinations. Let R_e be the configurations in which replay observes event e, and let P_e be the configurations in which a plan may request it. The observations determine the operator on P_e when

$$h|_{R_e} = h'|_{R_e} \quad \Longrightarrow \quad h|_{P_e} = h'|_{P_e} \qquad \text{for all } h, h' \in \mathcal{H}_{d,e}. \tag{14}$$

If replay fixes an operator on R_e, this condition fixes its predictions on P_e. Puzzle uses this route. After replay identifies a repeated effect class, one exact transition of a globally enabled XOR operator determines its mask as $m_e = r \oplus r'$. Held-out transitions then test whether the class and mask are stable. At execution time, the executor must still realize the requested event. Feedback reveals the realized register change, and the planner continues from that state.

The closed-loop result in proposition 1 formalizes this interaction. When each realized event decreases a nonnegative task rank, replanning reaches the goal within the initial rank. The appendix proves the compiler contract and determining-set result and gives finite-data and stochastic extensions.

2.4 Puzzle as an XOR special case

The Puzzle compiler searches for complementary binary coordinates that persist between switch events. It groups repeated register changes and compares identity, XOR, COPY/SET/RESET, and general Boolean operators on held-out transitions. XOR is selected on all eight datasets. For an accepted event, the observed binary difference becomes its mask, and replay supplies switch targets that the linker can present to the executor. A separate pre-effect compiler audit recovers the same unique XOR updates, up to relabeling and duplicate event labels.

Puzzle succeeds exactly when its current and goal registers agree. Its register operators are globally enabled ($G_e \equiv 1$), context independent, commuting, and involutive. Let $r, g \in \mathbb{F}_2^n$, so $\tau(r, g) = 1$ if and only if $r = g$. Event e_j has a learned mask m_j.

$$F_{e_j}(r, g) = (r \oplus m_j, g). \tag{15}$$

Let $M \in \mathbb{F}_2^{n \times m}$ contain the masks as columns. The goal residual is $q(r, g) = r \oplus g$.

Corollary 1 (Exact Puzzle planning and quotient size)**.** *The goal is reachable exactly when* $q(r, g) \in \mathrm{im}\, M$. *A minimum unit-cost event plan solves*

$$\min_{z \in \mathbb{F}_2^m} \|z\|_0 \quad \textit{subject to} \quad Mz = q(r, g) \pmod 2. \tag{16}$$

The residual maps 2^{2n} absolute start–goal pairs to 2^n goal-relative configurations for any M. If $\rho = \operatorname{rank} M$, the quotient among valid start–goal configurations has

$$2^\rho + \mathbf{1}[\rho < n] \tag{17}$$

classes, one class for each reachable residual and, when $\rho < n$, one class for all unreachable residuals. The totalized quotient $\mathcal{Q}^*$ always has $2^\rho + 1$ classes. When $\rho < n$, the artificial rejecting state joins the unreachable class because both have an empty continuation language.

The Puzzle implementation enumerates the null space exactly when its nullity is at most 22, which holds in the reported experiments. Its worst-case enumeration cost is $O(\mathrm{poly}(n, m)2^{m-\rho})$. The generic elimination shortcut used by other adapters returns a feasible solution rather than a minimum-cardinality event word.

3 Experiments

3.1 Questions and protocol

We ask three questions. Can the compiler recover reusable event rules from replay? Can their composition solve goals that search over recorded states misses? Does the same compile, plan, execute loop help outside Puzzle? We use OGBench (Park et al., 2025), with all eight Puzzle tasks as the main benchmark.

For Puzzle, we train eight GCIQL policies for one million updates. Each policy is evaluated on the five official tasks with 50 episodes per task. GCIQL, TTGS, and ATQ use the same policy, replay, initial state, and goal in every comparison. We use the released TTGS implementation and its default graph and search settings (Opryshko et al., 2026). Play uses $\alpha = 1.0$, and Noisy uses $\alpha = 0.03$.

The remaining environments have different motor interfaces, so they use executors suited to their local goal spaces. Maze and AntSoccer use LAVL (Kang et al., 2026), Cube uses event-conditioned GCBC or DQC (Li et al., 2026), Scene uses HIQL, and Powderworld applies discrete events directly. ATQ does not retrain these executors. Each Base–ATQ comparison uses the same executor parameters, observation and action interfaces, and evaluation tasks. The appendix gives the training and evaluation counts.

3.2 Puzzle

Table 1: Puzzle success (%). Base, +TTGS, and +ATQ share eight GCIQL policies and evaluation episodes (mean ± SEM). Other columns show reported means. Recent bests select among DQC, TRL, ORS, QHyer, and CTA (table 12). Bold marks the highest displayed mean per task.

	Original OGBench (Park et al., 2025)						Our evaluation			Recent papers
Task	GCBC	GCIVL	GCIQL	QRL	CRL	HIQL	Base	+TTGS	+ATQ	Best reported
3x3-play	2	6	95	1	3	12	95.7 ± 0.6	98.0 ± 0.6	**100.0** ± 0.0	TRL 99[a]
3x3-noisy	1	42	94	0	30	51	93.3 ± 1.0	98.0 ± 0.5	**100.0** ± 0.0	QHyer 89
4x4-play	0	13	26	0	0	7	25.3 ± 1.5	24.9 ± 1.6	**98.5** ± 0.5	CTA 84
4x4-noisy	0	20	29	0	0	16	38.1 ± 2.7	44.1 ± 1.7	95.3 ± 1.5	CTA **96**
4x5-play	0	7	14	0	1	4	14.2 ± 0.4	1.9 ± 0.6	**100.0** ± 0.0	TRL 97[a,b]
4x5-noisy	0	19	19	0	3	5	19.4 ± 0.1	2.3 ± 0.5	**98.0** ± 0.6	QHyer 26
4x6-play	0	10	12	0	4	3	12.5 ± 0.7	0.8 ± 0.2	**99.4** ± 0.2	DQC 83[a,b]
4x6-noisy	0	17	18	0	6	2	17.7 ± 0.4	0.2 ± 0.1	**98.9** ± 0.5	QHyer 24

[a]Oracle goal representation. [b]1B replay transitions. Our 4x5/4x6 runs use released 3M/5M replay. Published scores follow their source protocols.

ATQ reaches 95.3–100.0% on every task. GCIQL and TTGS also solve 3x3, but their performance falls as the board grows. On 4x4–4x6, TTGS reaches 0.2–44.1%, whereas ATQ reaches 95.3–100.0%.

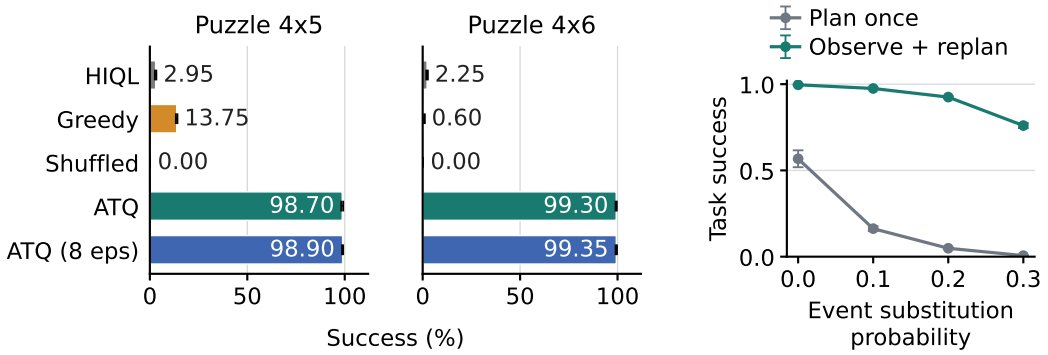

Figure 2: The task circuit is separate from the motor policy and is used in a closed loop. Left. ATQ composes events with eight independently trained HIQL policies on Puzzle 4x5 and 4x6. The final bar fits the event rules from only eight replay episodes. Right. Feedback preserves high success when requested events are randomly replaced, whereas executing the original plan fails. Error bars are SEM over policies.

ATQ improves all eight paired policies over GCIQL. It also improves every non-tied policy over TTGS. The paired intervals and sign counts are reported in table 8.

The table also gives the original OGBench baselines (Park et al., 2025) and the highest reported mean among five recent conference methods, DQC, TRL, ORS, QHyer, and CTA. ATQ exceeds these references on all four 4x5 and 4x6 tasks (100.0 versus 97, 98.0 versus 26, 99.4 versus 83, and 98.9 versus 24). The full method-by-task comparison, sources, and evaluation conditions appear in table 12.

Three tests distinguish operator composition from replay lookup. First, the compiled operators predict 99.914–100.000% of complete register changes on accepted event transitions in official validation replay. Second, a replay graph with more than 15,000 states connects only 3.5–5.0% of sampled 4x4 start–goal pairs. The compiled circuit plans all sampled pairs. Third, 98.4–100.0% of requests on 4x4–4x6 combine a register and event that never co-occur in compiler training data. Episodes in which every request is such a new combination still succeed 95.3–100.0% (table 11).

The controls support the same explanation. A one-step greedy planner performs poorly, and shuffling the correspondence between events and their effects reduces success to 0% on every task (table 9). With a second set of eight HIQL executors, ATQ reaches $98.70 \pm 0.51\%$ on 4x5 and $99.30 \pm 0.30\%$ on 4x6, compared with $2.95 \pm 0.36\%$ and $2.25 \pm 0.40\%$ without ATQ. These HIQL executors still realize 97.8–98.3% of completed shuffled requests, so local control alone does not explain success. Finally, when 20% of requested events are replaced by another event, replanning from feedback retains $92.50 \pm 0.53\%$ success. Executing the original plan without replanning yields $4.88 \pm 0.52\%$.

3.3 Beyond Puzzle

Table 2 reports the remaining 42 OGBench tasks in our study. The first six method columns reproduce the reference scores reported by OGBench (Park et al., 2025). For learned executors, the Base and +ATQ columns compare the same executor with and without ATQ under the same evaluation setting.

Table 2: Success (percent) on the 42 tasks beyond Puzzle. The six reference columns reproduce the original OGBench results (Park et al., 2025). Our results are mean ± SEM over eight policy seeds (compiler seeds for Powderworld). Base and +ATQ use the same executor named in each group. Bold marks the higher displayed mean within each matched pair.

	OGBench reference results						Our evaluation	
Task	GCBC	GCIVL	GCIQL	QRL	CRL	HIQL	Base	+ATQ
PointMaze (LAVL executor)								
medium navigate	9	63	53	82	29	79	89.9 ± 1.2	**94.1** ± 0.7
large navigate	29	45	34	86	39	58	94.3 ± 1.3	**96.0** ± 1.1
giant navigate	1	0	0	68	27	46	91.1 ± 2.9	**96.1** ± 1.5
teleport navigate	25	45	24	4	24	18	26.8 ± 1.4	**45.8** ± 3.3
medium stitch	23	70	21	80	0	74	87.8 ± 1.2	**92.5** ± 1.0
large stitch	7	12	31	84	0	13	86.4 ± 3.1	**94.0** ± 0.8
giant stitch	0	0	0	50	0	0	95.2 ± 0.5	**98.1** ± 0.7
teleport stitch	31	44	25	9	4	34	27.8 ± 3.6	**33.9** ± 2.5
AntMaze (LAVL executor)								
medium navigate	29	72	71	88	95	96	**96.9** ± 0.4	96.8 ± 0.4
large navigate	24	16	34	75	83	91	92.2 ± 0.4	**93.1** ± 0.7
giant navigate	0	0	0	14	16	65	**82.6** ± 1.0	81.7 ± 1.6
teleport navigate	26	39	35	35	53	42	**46.4** ± 2.0	45.1 ± 1.2
medium stitch	45	44	29	59	53	94	96.5 ± 0.3	96.5 ± 0.4
large stitch	3	18	7	18	11	67	**92.5** ± 0.7	91.8 ± 0.5
giant stitch	0	0	0	0	0	2	**78.5** ± 1.6	78.1 ± 1.6
teleport stitch	31	39	17	24	31	36	36.4 ± 1.3	**61.0** ± 1.4
medium explore	2	19	13	1	3	37	98.3 ± 0.4	**98.5** ± 0.3
large explore	0	10	0	0	0	4	**93.6** ± 0.8	93.4 ± 0.8
teleport explore	2	32	7	2	20	34	**36.3** ± 1.8	29.6 ± 1.7
HumanoidMaze (LAVL executor)								
medium navigate	8	24	27	21	60	89	94.2 ± 0.4	**94.9** ± 0.4
large navigate	1	2	2	5	24	49	77.8 ± 1.2	**81.8** ± 1.1
giant navigate	0	0	0	1	3	12	**84.0** ± 1.8	63.2 ± 1.4
medium stitch	29	12	12	18	36	88	92.0 ± 0.4	**94.4** ± 0.5
large stitch	6	1	0	3	4	28	73.3 ± 1.8	**77.6** ± 1.2
giant stitch	0	0	0	0	0	3	**82.0** ± 1.1	56.0 ± 1.4
AntSoccer (LAVL executor)								
arena navigate	5	47	50	8	23	58	8.9 ± 0.6	**29.6** ± 1.3
medium navigate	2	4	7	2	3	13	5.3 ± 0.4	**6.3** ± 0.6
arena stitch	24	21	2	1	1	15	2.1 ± 0.2	**12.5** ± 1.0
medium stitch	2	1	0	0	0	4	1.9 ± 0.4	**3.6** ± 0.6
Cube (event GCBC / DQC executors)								
single play	6	53	68	5	19	15	**0.1** ± 0.1	0.0 ± 0.0
double play	1	36	40	1	10	6	0.0 ± 0.0	**20.9** ± 1.0
triple play	1	1	3	0	4	3	0.1 ± 0.1	**27.3** ± 1.4
quadruple play†*	0	0	0	0	0	0	87.7 ± 0.6	87.7 ± 0.9
single noisy	8	71	99	25	38	41	0.7 ± 0.2	**1.1** ± 0.2
double noisy†	1	14	23	3	2	2	**22.9** ± 1.0	22.1 ± 1.3
triple noisy†	1	9	2	1	3	2	29.5 ± 3.0	**30.2** ± 3.2
quadruple noisy†	0	0	0	0	0	0	**9.4** ± 0.5	9.0 ± 0.7
Scene (HIQL executor)								
play	5	42	51	5	19	38	39.4 ± 2.2	**56.1** ± 2.5
noisy	1	26	26	9	1	25	25.7 ± 0.8	**39.3** ± 1.2
Powderworld (direct event execution)								
easy play	0	99	93	12	22	33	74.9 ± 7.1	**100.0** ± 0.0
medium play	1	50	16	3	1	22	N/A	59.6 ± 0.0
hard play	0	4	0	0	0	1	N/A	43.3 ± 0.1

Cube uses event GCBC except †DQC rows. *Quadruple play uses 100M replay and DQC's oracle-goal training representation.

ATQ gains more than one percentage point on 21 of the 40 tasks with a Base comparison. It stays within one point on 15 more. The gains cover every PointMaze and AntSoccer task and both Scene tasks. Notable gains among learned executors include 27.2 points on Cube triple play, 24.6 on AntMaze teleport stitch, 20.9 on Cube double play, 20.7 on AntSoccer arena navigate, 19.0 on PointMaze teleport navigate, and 16.7 on Scene play. Many small changes occur when Base already exceeds 90%. This pattern is consistent with ATQ helping most when local goal reaching is available but event ordering remains difficult.

ATQ is not uniformly beneficial. AntMaze teleport explore decreases by 6.7 points. HumanoidMaze giant navigate and giant stitch decrease by 20.8 and 26.1 points. The HumanoidMaze declines are consistent with a mismatch between planned events and the executor's reliable local goals.

3.4 Direct event execution in Powderworld

Powderworld tests a second algebra of events. Instead of toggling binary registers, each operator assigns a value to a local grid region. The compiler discovers repeated three-action codes and factorizes them into up to 512 sparse assignment operators. With direct event execution, ATQ reaches 100.0%, 59.6%, and $43.3 \pm 0.1\%$ on Easy, Medium, and Hard. Permuting the assignments reduces success to 0% on all three tasks. This result shows that the compiler is not restricted to XOR updates.

4 Related Work

Offline goal-conditioned RL commonly relabels future states as goals or learns a geometry for reaching them (Park et al., 2025; Hansen-Estruch et al., 2022). Recent methods improve stitching through hierarchical subgoals, temporal distances, or decision-time graph search (Park et al., 2023; Choi et al., 2026; Myers et al., 2025; Opryshko et al., 2026). ATQ instead exposes discrete event effects to a planner and delegates motor control to the policy. ALPS searches a learned representation graph at decision time (Shehmar et al., 2026). ATQ may use a replay graph to propose places, but its distinct object is a labeled update to a task register.

Learning symbolic models for planning has a long lineage. Action-model acquisition infers operators from plans (Yang et al., 2007). Skills-to-Symbols derives planning abstractions from given high-level actions, while LatPlan recovers latent propositions and actions from image transitions (Konidaris et al., 2018; Asai & Fukunaga, 2018). LOFT and NSRT learn symbolic operators that guide continuous robot planning (Silver et al., 2021; Chitnis et al., 2022). Predicate invention learns state features (Silver et al., 2023), while goal-relevant prediction avoids modeling irrelevant changes (Kumar et al., 2023). ATQ brings operator discovery to passive public goal-conditioned replay and connects the learned operators to an unchanged executor. Its interventions test whether the resulting event semantics, rather than additional policy training, explain long-horizon success.

State abstraction asks when two states may be merged without changing control (Givan et al., 2003; van der Pol et al., 2020). Group-structured models learn repeatable transformations from observed transitions (Keurti et al., 2023; Kim et al., 2026). Abstract world models derive value-preserving models from given options (Rodriguez-Sanchez & Konidaris, 2024). Reward machines encode temporal task structure (Toro Icarte et al., 2022; 2023). The ATQ quotient is a standard deterministic residual-language abstraction, related to general interaction quotients (Nixon, 2026). ATQ contributes the replay compiler contract, the stateful link to a learned executor, and experiments that test composition and feedback directly.

5 Conclusion

ATQ compiles repeated task changes from replay and composes them without retraining the executor. On all eight Puzzle tasks, this separation raises success to 95.3–100.0% and applies learned events in register contexts absent from compiler training data. The direct TTGS comparison shows that these gains do not follow from search over recorded states alone. The shuffled and feedback interventions further show the contribution of correct event semantics and closed-loop replanning.

The broader OGBench results identify the scope of the idea. ATQ improves local executors across navigation, manipulation, soccer, and spatial assignment. It helps most when the executor can realize a requested event but cannot order many events on its own. The declines on giant HumanoidMaze highlight the importance of matching events to reliable local requests. Thus the candidate library and linker are useful inductive biases rather than a universal control interface.

Reproducibility Statement

The appendix specifies the candidate libraries, data splits, executor for each family, planner settings, training budgets, evaluation counts, and uncertainty units. The anonymous supplement contains the compiler and Puzzle execution code, evaluation settings, per-policy results, learned Puzzle operators, and numerical results for the reported tables and figures.

AI Use Statement

In this work, we used generative AI tools to assist with research brainstorming, hypothesis refinement, theoretical model formulation, mathematical claim and proof drafting, experimental design, code implementation, data analysis, result interpretation, literature search, and manuscript drafting and editing. The authors take responsibility for the final content, including all text, claims, proofs, code, and artifacts produced with the aid of generative AI.

References

Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.12077.

Rohan Chitnis, Tom Silver, Joshua B. Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

Jinwoo Choi, Sang-Hyun Lee, and Seung-Woo Seo. Chain-of-goals hierarchical policy for long-horizon offline goal-conditioned reinforcement learning. *arXiv preprint arXiv:2602.03389*, 2026.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1–2):163–223, 2003. doi: 10.1016/S0004-3702(02)00376-4.

Philippe Hansen-Estruch, Amy Zhang, Ashvin Nair, Patrick Yin, and Sergey Levine. Bisimulation makes analogies in goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, 2022.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 3 edition, 2006.

Hyungkyu Kang, Byeongchan Kim, and Min-hwan Oh. Latent representation alignment for offline goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, 2026.

Hamza Keurti, Hsiao-Ru Pan, Michel Besserve, Benjamin F. Grewe, and Bernhard Schölkopf. Homomorphism AutoEncoder: Learning group structured representations from observed transitions. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 16190–16215, 2023.

Junseok Kim, Dohyeong Kim, Mineui Hong, and Songhwai Oh. Compositional transduction with latent analogies for offline goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, 2026.

George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018. doi: 10.1613/jair.5575.

Nishanth Kumar, Willie McClinton, Rohan Chitnis, Tom Silver, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning efficient abstract planning models that choose what to predict. In *Proceedings of the Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2070–2095, 2023.

Xing Lei, Jincheng Wang, Xuetao Zhang, and Donglin Wang. QHyer: Q-conditioned hybrid attention-mamba transformer for offline goal-conditioned RL. In *International Conference on Machine Learning*, 2026.

Qiyang Li, Seohong Park, and Sergey Levine. Decoupled Q-chunking. In *International Conference on Learning Representations*, 2026.

Vivek Myers, Bill Chunyuan Zheng, Benjamin Eysenbach, and Sergey Levine. Offline goal-conditioned reinforcement learning with quasimetric representations. *arXiv preprint arXiv:2509.20478*, 2025.

Anthony T. Nixon. The myhill–nerode theorem for bounded interaction: Canonical abstractions via agent-bounded indistinguishability. *arXiv preprint arXiv:2603.21399*, 2026.

Evgenii Opryshko, Junwei Quan, Claas Voelcker, Yilun Du, and Igor Gilitschenski. Test-time graph search for goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, 2026.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. HIQL: Offline goal-conditioned reinforcement learning with latent states as actions. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34866–34891, 2023.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations*, 2025.

Seohong Park, Aditya Oberai, Pranav Atreya, and Sergey Levine. Transitive RL: Value learning via divide and conquer. In *International Conference on Learning Representations*, 2026.

Rafael Rodriguez-Sanchez and George Konidaris. Learning abstract world models for value-preserving planning with options. *Reinforcement Learning Journal*, 4:1733–1758, 2024.

Dikshant Shehmar, Matthew Schlegel, Matthew E. Taylor, and Marlos C. Machado. Laplacian representations for decision-time planning. In *International Conference on Machine Learning*, 2026.

Tom Silver, Rohan Chitnis, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Learning symbolic operators for task and motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomas Lozano-Perez, Leslie Pack Kaelbling, and Joshua B. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 12120–12129, 2023. doi: 10.1609/aaai.v37i10.26429.

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. doi: 10.1016/j.artint.2023.103989.

Elise van der Pol, Daniel E. Worrall, Herke van Hoof, Frans A. Oliehoek, and Max Welling. MDP homomorphic networks: Group symmetries in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Aravind Venugopal, Jiayu Chen, Xudong Wu, Chongyi Zheng, Benjamin Eysenbach, and Jeff Schneider. Occupancy reward shaping: Improving credit assignment for offline goal-conditioned reinforcement learning. In *International Conference on Learning Representations*, 2026.

Hrishikesh Viswanath, Juanwu Lu, S. Talha Bukhari, Damon Conover, Ziran Wang, and Aniket Bera. Physics informed viscous value representations. *arXiv preprint arXiv:2602.23280v1*, 2026. Version 1. The Powderworld table is not present in version 2.

Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2–3):107–143, 2007.

A Formal Details

A.1 Exact compiler factorization

We prove theorem 1. Let $\mathcal{U}_0 = \phi(\mathcal{X})$. Assume eq. (2). For any $u \in \mathcal{U}_0$, choose $x \in \phi^{-1}(u)$ and define

$$\tau(u) = a(x), \qquad G_e(u) = \mathbf{1}[x \in D_e^{\mathcal{X}}], \qquad F_e(u) = \phi(T_e x) \text{ when } G_e(u) = 1. \tag{18}$$

Acceptance consistency makes τ independent of the representative x. Guard consistency does the same for G_e. Successor consistency makes F_e independent of the representative on its guarded domain. These functions are therefore well defined and satisfy eq. (3).

Conversely, suppose deterministic functions τ, G_e, and F_e satisfy eq. (3), with the successor equality required on applicable source records. Two records in one ϕ-fiber must have the same acceptance value and event-domain indicator. When an event is defined at both records, their successors map to the same $F_e(\phi(x))$. Thus all three lines of eq. (2) are necessary.

For trace preservation, induct on word length. The empty word is immediate. If $\phi(T_w x) = F_w(\phi(x))$ and the next source event e is applicable, then

$$\phi(T_e T_w x) = F_e(\phi(T_w x)) = F_e F_w(\phi(x)). \tag{19}$$

Totalize the source as in eq. (5), with absorbing acceptance and a rejecting sink mapped by ϕ to $\perp$. Guard consistency also makes unsupported extensions agree under this totalization. Acceptance consistency then preserves success after every word. Hence two records in one extractor fiber have the same successful continuation language. Equivalently, $\ker \phi$ refines source continuation equivalence.

A.2 Ideal quotient

We restate the complete deterministic construction. For $w = e_1 \cdots e_K$, define

$$\overline{F}_w = \overline{F}_{e_K} \circ \cdots \circ \overline{F}_{e_1}, \qquad \overline{F}_\epsilon = \mathrm{id}\,. \tag{20}$$

The totalization in eq. (5) makes this expression well defined for every word, including words that contain an unsupported event. Because acceptance is absorbing, a word is successful exactly when one of its prefixes first reaches the task goal.

If $u \equiv v$, then $\mathcal{L}_u = \mathcal{L}_v$. For any e and w,

$$w \in \mathcal{L}_{\overline{F}_e(u)} \iff ew \in \mathcal{L}_u \iff ew \in \mathcal{L}_v \iff w \in \mathcal{L}_{\overline{F}_e(v)}. \tag{21}$$

Thus continuation equivalence is a right congruence. The empty word proves that the goal predicate is constant inside each class. Equal languages have the same set of successful word lengths, so they have the same shortest unit-cost event distance, defined as $+\infty$ for an empty language. Finally, let $\approx$ be an equivalence that preserves success after every event word. If $u \approx v$, then $\overline{\tau}(\overline{F}_w(u)) = \overline{\tau}(\overline{F}_w(v))$ for all w. Hence $\mathcal{L}_u = \mathcal{L}_v$, so $\approx$ refines continuation equivalence. Consequently, δ^* and τ^* in eq. (7) are well defined.

This argument does not show that the learned circuit is minimal. When a finite reachable transition table is available, the compiler can minimize that table and compare the result with the retained circuit. Other adapters retain a tested refinement or approximation.

A.3 Conditional closed-loop correctness

At runtime, the linker and executor act outside the task-event circuit. For a request e they produce a target and updated request memory, $(y, \widetilde{m}) = \widehat{\Gamma}_{d,\theta}(e, H^o, g, m)$. After execution, the observer records either a realized event $\widetilde{e}$ or a timeout. The exact linked-execution condition for a realized event is

$$\widehat{\phi}(H^{o,+}, g) = \widehat{F}_{\widetilde{e}}(\widehat{\phi}(H^o, g)). \tag{22}$$

The event $\widetilde{e}$ may differ from the request if feedback identifies an alternative in the event bank.

Proposition 1 (Conditional closed-loop correctness)**.** *Let $\tau_{\mathrm{env}}(H, g)$ be the original task-success predicate. Suppose*

$$\tau_{\mathrm{env}}(H, g) = \widehat{\tau}\big(\widehat{\phi}(H, g)\big), \qquad \widehat{\mathcal{A}} = \{u : \widehat{\tau}(u) = 1\}. \tag{23}$$

At every closed-loop-reachable nonaccepting state, suppose the planner returns a nonempty word whose first event satisfies the circuit guard. On every closed-loop-reachable state–event pair, suppose the accepted operator predicts the extracted successor. Suppose each executor attempt realizes the requested event or an observed alternative in the event bank before timeout, and that the execution budget permits at least $V(u_0)$ attempts. Finally, suppose there is a rank $V : \widehat{\mathcal{U}} \to \mathbb{N}_0$ such that $V(u) = 0$ exactly on $\widehat{\mathcal{A}}$ and every realized nonaccepting event decreases V. Then Algorithm 1 reaches the original task goal in at most $V(u_0)$ event attempts.

Proof. Induct over event boundaries. Planner progress prevents Algorithm 1 from returning failure at a reachable nonaccepting state. Operator correctness and observation make the updated circuit state equal to the task configuration extracted from the realized history, including after an observed alternative. Replanning therefore starts from the realized state. The nonnegative integer rank decreases after each attempt, so it reaches zero within $V(u_0)$ attempts. Equation (23) then implies $\tau_{\mathrm{env}} = 1$. □

The proposition states a semantic condition for closed-loop correctness. The experiments measure its components but do not certify a decreasing rank for every realized outcome.

A.4 Replay evidence for planning

Replay can support planning through interpolation or through identification of a restricted event operator. These are different assumptions.

Support interpolation. Let $d_{\mathcal{D}}(u, e)$ be the replay distribution over supported configuration–event pairs. Let $d_{\mathrm{plan}}(u, e)$ be the occupancy induced by the planner and executor. Suppose

$$d_{\mathrm{plan}}(u, e) > 0 \Longrightarrow d_{\mathcal{D}}(u, e) > 0, \qquad \frac{d_{\mathrm{plan}}(u, e)}{d_{\mathcal{D}}(u, e)} \le C. \tag{24}$$

For every nonnegative operator loss,

$$R_{\mathrm{plan}}(\widehat{F}) = \mathbb{E}_{d_{\mathrm{plan}}}[\ell(\widehat{F}; u, e)] \le C\, \mathbb{E}_{d_{\mathcal{D}}}[\ell(\widehat{F}; u, e)] = C R_{\mathcal{D}}(\widehat{F}). \tag{25}$$

This is a standard covariate-shift argument. Passing exact pairwise replay-membership checks establishes support inclusion on the checked domain. Occurrence counts and learned guards alone do not certify this condition or bound C.

For a failure-probability statement, specialize to the zero–one operator loss $\ell(\widehat{F}; u, e) = \mathbf{1}[\widehat{F}_e(u) \neq F_e(u)]$ and let $\varepsilon_{\mathrm{op}} = R_{\mathcal{D}}(\widehat{F})$. Suppose compiler selection fails with probability at most δ_{sel}. Conditional on successful selection, suppose every error-free closed-loop execution reaches acceptance within K attempts. Let $\mathcal{F}$ denote any operator, executor, or observation error before acceptance. Conditional on successful selection and no earlier error, bound these three error probabilities by $C\varepsilon_{\mathrm{op}}$, $\varepsilon_{\mathrm{exec}}$, and $\varepsilon_{\mathrm{obs}}$. A union bound gives

$$\Pr(\mathcal{F}) \le \min\left\{1, \delta_{\mathrm{sel}} + K\left(C\varepsilon_{\mathrm{op}} + \varepsilon_{\mathrm{exec}} + \varepsilon_{\mathrm{obs}}\right)\right\}. \tag{26}$$

No independence between attempts is required. Each conditional bound must hold under histories produced by earlier error-free attempts. This expression excludes the absence of a valid plan, cycling, and horizon termination. A numerical guarantee additionally requires estimates of every error term.

Structural extrapolation. For event e, let R_e be the replay source configurations and P_e the configurations where the planner may apply the event. Let $\mathcal{H}_e$ be the declared operator family.

Proposition 2 (Determining replay set)**.** *Suppose the true operator h_e^* and selected operator $\widehat{h}_e$ belong to $\mathcal{H}_e$. Suppose $\widehat{h}_e|_{R_e} = h_e^*|_{R_e}$ and R_e determines $\mathcal{H}_e$ on P_e as in eq. (14). Then $\widehat{h}_e|_{P_e} = h_e^*|_{P_e}$ even if some pairs in $P_e \times \{e\}$ never occur in replay.*

Proof. Apply eq. (14) to $\widehat{h}_e$ and h_e^*. □

For a globally enabled XOR family $\mathcal{H}_e = \{r \mapsto r \oplus m : m \in \mathbb{F}_2^n\}$, one exact labeled transition determines the mask because $m = r \oplus r'$. This statement assumes that the effect class is already correct and that the XOR family is valid globally. The Puzzle compiler uses repeated effects and independent validation to test stability under noise. Event-type coverage in eq. (36) supplies the corresponding replay requirement.

A.5 Finite-library selection

Let $\xi_1, \dots, \xi_B \overset{\text{iid}}{\sim} P_{\text{dev}}$. Let $\ell(h;\xi) \in [0,1]$ be a per-episode prediction loss, and define $R(h) = \mathbb{E}_{\xi\sim P_{\text{dev}}}[\ell(h;\xi)]$. Conditional on the fit set, assume that $\mathcal{H}_{\text{fit}}$ is finite and independent of the development episodes. Define

$$\widehat{R}_B(h) = \frac{1}{B}\sum_{i=1}^{B} \ell(h;\xi_i), \qquad \varepsilon_B(\delta) = \sqrt{\frac{\log(2|\mathcal{H}_{\text{fit}}|/\delta)}{2B}}. \tag{27}$$

Hoeffding's inequality and a union bound imply, with probability at least $1-\delta$,

$$\sup_{h\in\mathcal{H}_{\text{fit}}} |\widehat{R}_B(h) - R(h)| \le \varepsilon_B(\delta). \tag{28}$$

For the actual selector, define its empirical-risk sacrifice

$$\Delta_{\text{dev}} = \widehat{R}_B(\widehat{h}) - \min_{h\in\mathcal{H}_{\text{fit}}} \widehat{R}_B(h). \tag{29}$$

On the same event,

$$R(\widehat{h}) - \min_{h\in\mathcal{H}_{\text{fit}}} R(h) \le 2\varepsilon_B(\delta) + \Delta_{\text{dev}}. \tag{30}$$

The bound applies to empirical risk. The compiler splits complete episodes but often reports transition-weighted accuracy. We therefore do not insert transition counts into eq. (28). For predetermined episode weights $a_i \ge 0$ with $\sum_i a_i = 1$, weighted Hoeffding uses the effective episode count $B_{\text{eff}} = 1/\sum_i a_i^2$ under the same independence assumptions.

A.6 Stochastic event models

Language equivalence preserves possible deterministic continuations. It does not preserve stochastic success probabilities. For a finite or countable task configuration space, let P_e be a totalized event kernel on $\overline{\mathcal{U}}$, and let Π partition $\overline{\mathcal{U}}$. Strong lumpability requires that the goal predicate be constant inside each block and that

$$\sum_{x\in B} P_e(x \mid u) = \sum_{x\in B} P_e(x \mid v) \tag{31}$$

for every target block B, event e, and u, v in the same source block. The quotient kernel

$$P_e^{\Pi}(B' \mid B) = \sum_{x\in B'} P_e(x \mid u), \qquad u \in B, \tag{32}$$

is then well defined and preserves finite-horizon success probabilities for event policies that observe only quotient blocks. This is the stochastic extension of the deterministic contract. The experiments measure transition prediction but do not test strong lumpability directly.

A.7 Replay identifiability

Let

$$\Omega_{\mathcal{D}} = \{(u,e) : (u,e) \text{ has replay support}\}. \tag{33}$$

Two operator banks are observationally equivalent on replay when

$$h \equiv_{\mathcal{D}} h' \iff h_e(u) = h'_e(u) \quad \text{for every } (u,e) \in \Omega_{\mathcal{D}}. \tag{34}$$

A replay-only selector cannot distinguish candidates that induce the same distribution over observed replay variables. Such candidates may disagree outside $\Omega_{\mathcal{D}}$. Prediction beyond support therefore requires a restricted operator family and a determining replay set such as eq. (14).

B Compiler Language and Information Boundaries

B.1 Inputs to each compiler

When possible, the compiler identifies an event before reading its effect. The successor register then serves only as the target used to fit the operator. Puzzle groups repeated register changes, and Maze and Soccer define an event as a transition between replay-derived places. These event definitions therefore use both sides of a replay transition.

Table 3: Replay quantities used by each compiler. The candidate extractor and operator families are specified before fitting. Rewards, evaluator success, and simulator-private state are not compiler inputs.

Family	Candidate factors	Event evidence	Outcome blind	Accepted operator
Puzzle	repeated binary coordinates	observed register change in fit replay	no	XOR translation
Cube/Scene	repeated object registers	source, goal, segment, robot context	yes	COPY/SET/RESET
Powderworld	replay palette grid	repeated three-action code	yes	sparse assignment
Maze/Soccer	replay-continuous places	source and successor endpoint	no	guarded place transition

The factorization contract separates the frontend, task-event circuit, planner, and linker. The extractor $\widehat{\phi}$ forms a task configuration. The identifier $\widehat{\psi}$ names an event. The bank $(\widehat{F}, \widehat{G})$ predicts guarded task changes. The success test $\widehat{\tau}$ compares the selected task representation with the public goal. These objects form the task-event circuit. The family planner composes guarded events. The separate linker $\widehat{\Gamma}_{d,\theta}$ converts one request into an executor target using the current runtime context and memory.

Table 4: Family-specific compiler interfaces. *Replay populated* means that the adapter fixes the target type but replay determines its prototypes, locations, or action codes. The success test is not selected from reward or evaluator labels.

Family	$\widehat{\phi}$	$\widehat{\psi}$	$(\widehat{F}, \widehat{G})$	$\widehat{\tau}$	Linker target and context
Puzzle	repeated complementary binary coordinates	fit-split register-delta class	XOR translation with an occurrence support count	equality of selected current and goal bits	replay-populated event target prototype with current observation and retry memory
Cube/Scene	repeated object registers and persistent slow factors	pre-outcome segment classifier	COPY/SET/RESET or identity with learned support patterns	public goal comparison	replay-populated object endpoint or supported sequence with current object context
Maze/Soccer	replay-continuous places and route regions	source–successor place transition	bounded place update with replay-support guard	public goal-region test	replay-populated waypoint or transition endpoint with route and stall context
Powderworld	palette grid, address, and value factors	repeated three-action code	sparse assignment with radius/support guard	target-grid equality (planning)	replay-populated address/value action code with current grid

The compiler language includes complementary binary coordinate pairs, persistent two-mode scalars, repeated entity blocks, mixed slow factors, and sparse spatial addresses. Applicable event-operator candidates include identity, XOR translation, COPY/SET/RESET, general Boolean updates, sparse assignments, bounded translations, and guarded place transitions. Adapter-specific search bounds are human specified. The grounded Boolean compiler, for example, searches event counts 2 through 32 and temporal offsets 0 and 1.

The Puzzle event bank uses 60% of complete episodes for fitting and 40% for transition tests. A separate compiler test uses a 60/20/20 fit/development/test split to ask whether pre-effect evidence predicts event identity. Entity compilers use the same inner split, and Powderworld uses a 60/40 fit/test split. Official Puzzle validation replay measures operator accuracy and planning coverage but does not select candidates.

The full-replay Puzzle compiler requires at least 10 observations per event. It uses an error-band tolerance of 5×10^{-4}, a confidence threshold of 0.98, at most six full-law finalists, planning depth 64,

and 100,000 maximum expansions. Maze requires 20 observations of a place transition. Powderworld searches radii 0, 1, and 2 with minimum support 20, planning horizon 64, and at most 128 assignment steps. Entity confidence thresholds are selected on development episodes.

Each adapter maps a symbolic event to the target format expected by that family's executor. Puzzle requests a replay-supported switch effect. Cube requests a local object event endpoint. Scene requests a supported event sequence. Maze and Soccer request replay-supported locomotion endpoints. These targets are not a shared primitive action space. The shared object is the task-event circuit and its closed-loop composition.

B.2 Execution protocols

Table 5: Protocols for learned executors. Puzzle policies are trained for one million updates and evaluated on five official tasks with 50 episodes per task. Policy counts for each environment appear in tables 1 and 2.

Family	Executor	Policies	Compared methods	Feedback rule
Puzzle	GCIQL	8	controller alone, Greedy, TTGS, shuffled, ATQ	register change or 50-step timeout
Cube	event GCBC / DQC	8	controller alone, shuffled, ATQ	observed local object event or timeout
Scene	HIQL	8	controller alone, shuffled, ATQ	observed supported sequence or timeout
Maze/ AntSoccer	LAVL	8	controller alone, shuffled, ATQ	route progress, stall, or episode end

The compared methods in each row reuse the same trained policies, observation and primitive-action interfaces, and evaluation tasks. PointMaze teleport navigate seeds the environment, action-space, and global NumPy generators identically across methods for each evaluation episode. The Greedy Puzzle arm chooses the event with the smallest one-step Hamming residual. It does not solve for an event sequence.

The Cube DQC executors use 2.5 million updates for double noisy and quadruple noisy, and one million for triple noisy and quadruple play. Each is evaluated on five official tasks with 50 episodes per task.

The Puzzle TTGS comparison uses the released implementation. We retain its defaults $M = 4000$, $\tau = 24$, and $T = 48$, with dynamic penalty 1000, the adaptive farthest reachable subgoal, and no replanning. TTGS and ATQ load the same GCIQL policy and released replay. They also use the same task and episode starts. Puzzle samples five settling actions from a newly constructed action space during reset, so both evaluators initialize that action space identically. The TTGS paper reports 4x5-play and 4x6-play with its own policies and evaluation protocol. Our experiment extends the direct comparison to all eight OGBench Puzzle environments using the same trained policies and initial conditions for each method.

Puzzle permits 50 primitive steps for one requested event. On timeout, the wrapper records the failure, temporarily defers that event, and replans from the observed register. If every candidate event has been deferred, it clears the deferral set and retries. A realized register change, including a supported alternative event, immediately clears stale deferrals and triggers replanning.

B.3 Shuffled-circuit intervention

The shuffled control does not change the learned executor or the compiled set of local targets. It replaces the correspondence between an event symbol and its claimed effect or target with one permutation. The permutation is held constant across tasks and episodes. It has no fixed points when an informative derangement exists.

We use the shuffled control only when at least two informative event labels permit a nontrivial permutation. Puzzle also uses label shuffles when testing operator prediction. That test occurs before control and is distinct from the execution intervention above.

Table 6: The correspondence broken by the shuffled-circuit control. The control does not retrain or replace the executor.

Family	Preserved	Permuted correspondence
Puzzle	switch-effect bank, event targets, executor	event identity to claimed XOR effect
Cube	object-event targets and event GCBC executor	event identity to object effect target
Scene	supported event-sequence targets and HIQL executor	sequence identity to supported target
Maze/AntSoccer	replay-supported waypoint targets and LAVL executor	place transition to waypoint or transition target
Powderworld	sparse assignment operators and direct actuator	event identity to address/value assignment

B.4 Planner implementations

The general finite circuit planner uses unit-cost breadth-first search. A returned word is shortest only when the search terminates before its resource cap. Ordered irreversible tasks retain an uncompleted instruction suffix. Sparse assignment planners favor residual-reducing operators and allow plateau moves. These states are learned circuit states and are not claimed to be minimum quotient classes.

Puzzle uses the exact minimum-weight solver in eq. (16). The generic GF(2) elimination shortcut sets free variables to zero and returns only a feasible solution. For example,

$$M = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \qquad q = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{35}$$

The generic shortcut returns $z = (1, 1, 0)$ with weight two, while the exact minimum is $z = (0, 0, 1)$ with weight one.

C Puzzle Audits

Table 7: Structural audit of replay-compiled Puzzle circuits. P/N denotes play/noisy. Valid/total gives quotient classes among valid configurations and in the totalized space, respectively. Both are computed exactly from the accepted XOR bank. Operator accuracy is exact whole-register accuracy on accepted event transitions in official validation replay. The replay graph contains only observed transitions from the 60% fit episodes. Planning evaluates 200 reachable start-goal pairs per dataset from separate official validation replay (within episodes for 4x4). A dash means that the absolute replay graph was not enumerated at that size.

Size	Events P/N	Rank	Valid/total classes	Operator acc. P/N	Replay nodes P/N	Replay BFS P/N	ATQ plan P/N
3×3	9/9	9	512/513	100.000/100.000	512/512	100.0/100.0	100.0/100.0
4×4	16/21	12	4,097/4,097	100.000/99.914	15,293/16,736	3.5/5.0	100.0/100.0
4×5	20/25	20	1,048,576/1,048,577	100.000/99.921	–	–	100.0/100.0
4×6	24/24	24	16,777,216/16,777,217	100.000/99.962	–	–	100.0/100.0

Across 5,000 uniformly sampled reachable residuals per Puzzle row, the median Python time for the exact solver is 58–196 μs. The timing excludes environment execution and linker calls.

C.1 Selecting the Puzzle operators

The operator test runs separately on all eight OGBench Puzzle datasets, 3x3, 4x4, 4x5, and 4x6, each with play and noisy replay. It compares identity, XOR translation, goal-relative COPY/SET/RESET, and a general coordinate-wise Boolean operator. XOR is selected on every dataset. Exact prediction on official validation replay ranges from 0.99914 to 1.00000.

On 4x6, the event alphabet is clustered from source observations and action segments before candidate operators read the successor register. The general Boolean candidate independently reduces every output to coordinate-wise COPY/NOT wiring. Twenty frequency-preserving event-label shuffles obtain zero exact-transition accuracy. Thus replay selects the XOR bank from the candidate library rather than receiving the masks or event correspondence as input.

Table 8: Paired Puzzle statistics. Differences are percentage points. Each interval is a paired 95% bootstrap confidence interval from 10,000 seed resamples. Positive counts exclude exact ties. Each p is an unadjusted two-sided exact sign test.

	ATQ minus GCIQL				ATQ minus TTGS			
Environment	Δ	95% CI	Positive	Sign p	Δ	95% CI	Positive	Sign p
3x3-play	4.3	[3.2, 5.7]	8/8	0.0078	2.0	[1.0, 3.1]	7/7	0.0156
3x3-noisy	6.6	[5.0, 8.6]	8/8	0.0078	1.9	[1.1, 2.9]	8/8	0.0078
4x4-play	73.2	[69.5, 76.4]	8/8	0.0078	73.6	[70.8, 76.4]	8/8	0.0078
4x4-noisy	57.2	[51.6, 62.5]	8/8	0.0078	51.2	[48.8, 53.6]	8/8	0.0078
4x5-play	85.8	[85.0, 86.5]	8/8	0.0078	98.1	[96.9, 99.0]	8/8	0.0078
4x5-noisy	78.6	[77.6, 79.6]	8/8	0.0078	95.7	[94.2, 97.2]	8/8	0.0078
4x6-play	86.9	[85.5, 88.1]	8/8	0.0078	98.6	[98.1, 99.3]	8/8	0.0078
4x6-noisy	81.2	[79.9, 82.5]	8/8	0.0078	98.7	[97.6, 99.6]	8/8	0.0078

Table 9: Puzzle mechanism controls. Greedy chooses the event with the largest immediate reduction in Hamming distance. Shuffled preserves the policy and local event requests but permutes which task change each event is claimed to produce. Values are mean $\pm$ SEM (percent) over eight policies.

Task	Greedy	Shuffled	ATQ
3x3-play	10.5 ± 0.2	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$
3x3-noisy	9.8 ± 0.1	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$
4x4-play	2.4 ± 0.6	0.0 ± 0.0	$\mathbf{98.5 \pm 0.5}$
4x4-noisy	0.0 ± 0.0	0.0 ± 0.0	$\mathbf{95.3 \pm 1.5}$
4x5-play	14.9 ± 0.6	0.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$
4x5-noisy	19.8 ± 0.1	0.0 ± 0.0	$\mathbf{98.0 \pm 0.6}$
4x6-play	1.1 ± 1.0	0.0 ± 0.0	$\mathbf{99.4 \pm 0.2}$
4x6-noisy	0.0 ± 0.0	0.0 ± 0.0	$\mathbf{98.9 \pm 0.5}$

A separate context test keeps the accepted event identities and effect bank unchanged. It forms 20 deterministic episode partitions and evaluates event effects on held-out register–event pairs that are absent from each fit partition. The partitions reuse one replay pool and one accepted effect bank. They are not 20 independent operator discoveries, so we treat their variation as a descriptive sensitivity check rather than sampling uncertainty. Mean exact effect accuracy on unseen pairs is 99.98–100.00% across the eight datasets. The mean unseen pair fraction is 0.79–1.91% on 3x3 and 98.11–99.98% on the larger layouts. The low 3x3 fraction reflects saturation of its small finite state space.

C.2 Puzzle statistics and controls

The paired confidence intervals resample the eight independently trained policies 10,000 times. The exact sign tests use only the sign of the paired ATQ-minus-comparator difference. They are reported without a multiple-comparison adjustment. A Holm correction across the eight environments gives 0.0625 for each comparator family, so we use the tests descriptively and do not claim row-wise significance.

C.3 Composition outside observed register–event pairs

The context audit above evaluates the accepted operator bank on deterministic replay partitions. We also audit the requests that ATQ issues during the primary closed-loop evaluation. The compiler fit split defines whether a pair is seen. We compare the full source register vector and requested event identifier. This criterion does not label a pair unseen merely because its absolute simulator state differs.

The table pools episode and request counts. These values are descriptive and do not form additional independent policy seeds. The all-unseen rows show that the accepted XOR operators remain useful when the planner composes them in source register contexts absent from compiler fit replay. This test concerns the selected finite XOR family. It does not imply unrestricted extrapolation.

Table 10: TTGS runtime diagnostics. Times are wall-clock means across eight policies on NVIDIA H200 GPUs. Graph construction occurs once before evaluation. Shortest-path time is the mean per-episode initialization time. Subgoal time is the episode-averaged mean per decision.

Environment	Nodes	Build (s)	Shortest paths (s)	Subgoal (ms)	Infinite paths (%)
3x3-play	4,000	26.58	3.16	0.221	0.0
3x3-noisy	4,000	17.57	2.43	0.139	0.0
4x4-play	4,000	21.08	1.46	0.246	0.0
4x4-noisy	4,000	26.29	1.43	0.214	0.0
4x5-play	4,000	14.63	0.86	0.143	0.0
4x5-noisy	4,000	23.97	1.59	0.187	0.0
4x6-play	4,000	14.87	1.02	0.174	0.0
4x6-noisy	4,000	17.10	1.18	0.200	0.0

Table 11: Execution on event contexts absent from compiler fit replay. A request is unseen when its source register vector and requested event never occur together in the fit split. An all-unseen episode contains no seen request. Success is pooled across eight policy seeds and 2,000 episodes per environment.

Environment	Unseen requests (%)	All-unseen episodes	All-unseen success (%)	Overall success (%)
4x4-play	98.94	1,885	98.62	98.55
4x4-noisy	98.40	1,779	95.33	95.35
4x5-play	99.63	1,901	100.00	100.00
4x5-noisy	99.81	1,935	97.93	98.00
4x6-play	99.99	1,998	99.40	99.40
4x6-noisy	99.99	1,999	98.90	98.90

C.4 Comparison with published Puzzle results

The original baselines are from Park et al. (2025). The recent-method comparison includes DQC (Li et al., 2026), TRL (Park et al., 2026), ORS (Venugopal et al., 2026), QHyer (Lei et al., 2026), and CTA (Kim et al., 2026). Their values follow each source's training and evaluation protocol. DQC and TRL use oracle goal representations. The replay-size differences are marked in the table. Our GCIQL, TTGS, and ATQ rows use the matched policies and episodes described above. The accompanying `puzzle_published_results.json` records the source tables and reported uncertainty for each recent-method result.

C.5 Event-type coverage and replay efficiency

Suppose independent replay episode i contains event type e with probability $p_e \geq p_{\min} > 0$. The probability that B episodes miss at least one of $|\mathcal{E}|$ event types is at most

$$|\mathcal{E}| \exp(-Bp_{\min}). \tag{36}$$

For a given replay of N episodes in which e occurs in c_e episodes, a uniform subset of size B misses it with exact marginal probability

$$\frac{\binom{N-c_e}{B}}{\binom{N}{B}}. \tag{37}$$

A union bound gives

$$\Pr(\text{all event types covered}) \geq \left[1 - \sum_{e \in \mathcal{E}} \frac{\binom{N-c_e}{B}}{\binom{N}{B}}\right]_+. \tag{38}$$

Event-type coverage is weaker than planned state–event coverage.

Each random draw contains eight complete replay episodes, or about 8,000 transitions. Across 256 draws, the compiler recovers the complete accepted 20- and 24-event operator banks in 255/256 and 256/256 draws. Draws come from the same replay pool and can overlap. The Wilson 95% lower bounds are 0.978 and 0.985. The full released replay remains available to train the executor.

Table 12: Full Puzzle comparison (success, %). Prior work shows reported means. Our matched evaluation shows mean ± SEM over eight GCIQL policies. A dash denotes an unreported task. Bold marks the highest displayed mean per column.

	Play				Noisy			
Method	3×3	4×4	4×5	4×6	3×3	4×4	4×5	4×6
Original OGBench (Park et al., 2025)								
GCBC	2	0	0	0	1	0	0	0
GCIVL	6	13	7	10	42	20	19	17
GCIQL	95	26	14	12	94	29	19	18
QRL	1	0	0	0	0	0	0	0
CRL	3	0	1	4	30	0	3	6
HIQL	12	7	4	3	51	16	5	2
Recent conference papers								
DQC[a] (Li et al., 2026)	—	—	96	83[b]	—	—	—	—
TRL[a] (Park et al., 2026)	99	34	97[b]	51[b]	—	—	—	—
ORS (Venugopal et al., 2026)	—	70	20	20	—	56	—	19
QHyer (Lei et al., 2026)	92	28	31	18	89	33	26	24
CTA (Kim et al., 2026)	94	84	17	12	42	**96**	—	—
Our matched evaluation (8 policies)								
GCIQL	95.7 ± 0.6	25.3 ± 1.5	14.2 ± 0.4	12.5 ± 0.7	93.3 ± 1.0	38.1 ± 2.7	19.4 ± 0.1	17.7 ± 0.4
GCIQL + TTGS (Opryshko et al., 2026)	98.0 ± 0.6	24.9 ± 1.6	1.9 ± 0.6	0.8 ± 0.2	98.0 ± 0.5	44.1 ± 1.7	2.3 ± 0.5	0.2 ± 0.1
GCIQL + ATQ	**100.0** ± 0.0	**98.5** ± 0.5	**100.0** ± 0.0	**99.4** ± 0.2	**100.0** ± 0.0	95.3 ± 1.5	**98.0** ± 0.6	**98.9** ± 0.5

[a]Oracle goal representation. [b]1B replay transitions. Our 4x5/4x6 runs use released 3M/5M replay. Published scores follow their source protocols.

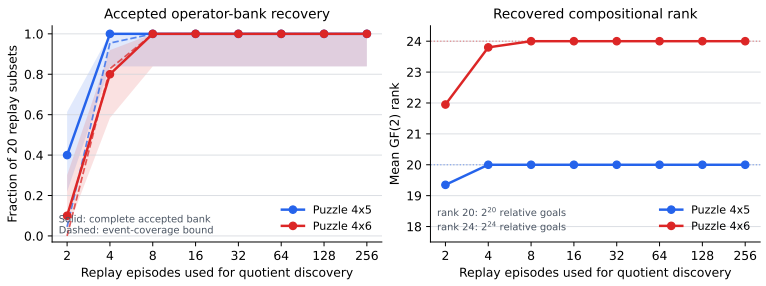

Figure 3: Recovery of the accepted Puzzle operator bank as replay episodes are subsampled. Dashed curves are the finite-replay event-type coverage bound. They do not certify register discovery or planned-pair coverage.

C.6 Proof of the XOR corollary and redundant operators

For a word w, let $\nu(w) \in \mathbb{F}_2^m$ record the parity of each event count. Before acceptance, commutativity and involution give, by induction on word length, the raw register update

$$F_w(r, g) = \big(r \oplus M\nu(w), g\big). \tag{39}$$

If a word succeeds under the absorbing semantics, its first successful prefix p satisfies $M\nu(p) = r \oplus g$. Conversely, any binary solution z defines a word that applies each selected event once and reaches acceptance no later than its final event. The minimum successful word length is therefore bounded below and above by the minimum Hamming weight in eq. (16), so the two are equal. This prefix argument is necessary because events written after the first successful prefix have no effect under the totalized semantics.

To derive eq. (17), let $A = \operatorname{im} M$ and fix a basis of A drawn from columns of M. Every residual outside A remains outside A after any event word, so its successful continuation language is empty. All such residuals therefore merge into one rejecting class. Now take distinct reachable residuals $q_1, q_2 \in A$. If either residual is zero, the empty word distinguishes them. Otherwise, write each residual in the chosen basis. Apply the basis events in the support of q_1 to cancel q_1. If the support of q_2 is not contained in that of q_1, no prefix can cancel q_2. If it is a proper subset, apply an event outside the support of q_2 first. Every nonempty prefix then contains that event and cannot equal the basis representation of q_2. The constructed word therefore succeeds from q_1 but not from q_2. Every reachable residual has its own continuation class. There are 2^ρ such residuals and one additional valid-configuration class exactly when $\rho < n$. This proves eq. (17). The artificial state $\perp$ has an empty continuation language. It joins the unreachable-residual class when $\rho < n$ and forms a new class when $\rho = n$. The full totalized quotient therefore has $2^\rho + 1$ classes in both cases.

Let $M \in \mathbb{F}_2^{n \times m}$ have rank ρ. If $Mz_0 = q$, every binary event-selection vector with the same effect is

$$\{z : Mz = q\} = z_0 + \ker M, \qquad |\{z : Mz = q\}| = 2^{m-\rho}. \tag{40}$$

Under globally enabled, commuting, involutive operators and unit event cost, a minimum-Hamming solution is a shortest represented event word. The 4x4 execution bank has 16 unique effects and rank 12, leaving 16 canonical binary event-selection vectors per reachable residual. Different orderings of one vector are not counted as different vectors. Unrestricted words can also insert cancelling event pairs, so eq. (40) is not a count of all event strings.

C.7 Feedback intervention

Pooled exact event agreement for HIQL/GCIQL on 4x5/4x6 play is 91.0–94.1%. Task recovery after a naturally occurring supported alternative is 97.9–100%. A controlled intervention gives the stronger comparison below.

The substitution experiment uses Puzzle 4x6 play and eight independently trained HIQL policies. Each policy is evaluated on all five official tasks with 20 episodes per task and condition. Each requested event is independently replaced with probability 0.2. The right panel of fig. 2 reports the result.

D POWDERWORLD DETAILS

Powderworld uses a direct event actuator instead of a learned executor. The planner searches the compiled circuit, and the actuator applies each requested assignment. The compiler factorizes repeated three-action codes into 64 replay addresses and their values, yielding 128, 320, and 512 sparse assignment operators on Easy, Medium, and Hard, respectively. It receives neither material names nor evaluator task sequences.

Table 13: Powderworld success under the standard OGBench evaluator (percent), mean $\pm$ SEM over eight compiler/evaluation runs, each evaluated for 500 episodes per arm. Easy Base uses open-loop execution of the compiled circuit. Medium and Hard have no controller-alone arm. Public values from Viscous Value v1 are contextual (Viswanath et al., 2026).

Task	Base	ATQ	Shuffled	Public
Easy	74.9 ± 7.1	100.0 ± 0.0	0.0 ± 0.0	99.0
Medium	N/A	59.6 ± 0.0	0.0 ± 0.0	62.0
Hard	N/A	43.3 ± 0.1	0.0 ± 0.0	10.0

On Medium, success for the five official tasks is 98%, 100%, 100%, 0%, and 0%, respectively, in each compiler run.

A separate five-split Hard audit recovers the same 512 assignment identifiers in every split. Held-out cell accuracy ranges from 79.4% to 83.5%. Failures occur after some learned assignments stop reducing the number of incorrect cells. The experiment does not isolate whether the remaining error comes from operator prediction, support shift, feedback, or event execution.