MULTI-LEVEL SEMANTIC SKILLS FOR TRANSFER: EXPLORATION, CONTROL, AND ADAPTATION ACROSS DISCRETE AND CONTINUOUS DOMAINS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hierarchical methods in reinforcement learning (RL) offer a natural substrate for transfer: a skill can keep the same semantic role even when the state representation, action space, or surrounding task changes. We exploit this principle by adapting a recently introduced multi-level RL framework, where learned policies become higher-level actions and can be reused through task-specific implementations. We develop three progressively broader forms of transfer. First, semantically meaningful skills learned in one task are reused to guide exploration and accelerate learning in related tasks. Second, hierarchical knowledge is carried from discrete state and action spaces to continuous states and continuous control by replacing lower-level implementations while preserving higher-level semantic structure. Third, learned hierarchical relationships are adapted to apparently different domains by separating reusable semantic structure from target-specific choices, including recursive decompositions. Experiments across navigation, control, and structured computation demonstrate the resulting gains in learning and adaptation when semantic structure is shared. Because these skills admit natural-language descriptions, they also provide an interpretable interface for communication, coordination, and cooperation in agentic systems.

1 INTRODUCTION

A useful skill can often be reused because of what it means, even when the way it is implemented changes (Konidaris et al., 2012). For example, an agent in a grid world and a continuous-control agent may use very different low-level actions, yet both may need to make the same high-level decision: "navigate to the goal." Likewise, Merge Sort and matrix-chain multiplication are different computational problems, but both can be organized around a common divide-and-conquer idea: solve smaller subproblems and then combine their solutions. This paper asks whether reinforcement learning (RL) can exploit such common structure: *can a learned high-level decision be transferred when the task, state representation, or low-level control changes?*

Our approach represents higher-level decisions through their semantic meaning. A *semantic skill* is a decision with a clear task-level interpretation, such as `Navigate`, `Serve`, or `SolveChild`. A source task learns when to use these skills. In a new target task, the same skill may be implemented in a different way. For example, `Navigate` may correspond to discrete moves in one environment and to a continuous controller in another. We build on a recently introduced multi-level RL framework (Yang & Maggioni, 2026), in which learned policies can themselves be used as higher-level actions. We assume that the correspondence between high-level skills in the source and target tasks is known, and study whether the learned policy over these skills can be transferred effectively.

We study three progressively broader forms of transfer.

- **Transfer across related tasks.** Semantically meaningful skills learned in one task are reused to guide exploration and accelerate learning in related tasks. For example, a policy that learns to navigate to a required object, retrieve it, and then navigate to the final goal can be transferred from MazeBase+ to Taxi. We also study a multi-request Taxi problem, where a learned `Serve` skill is reused while the target learns which request to serve next.

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.

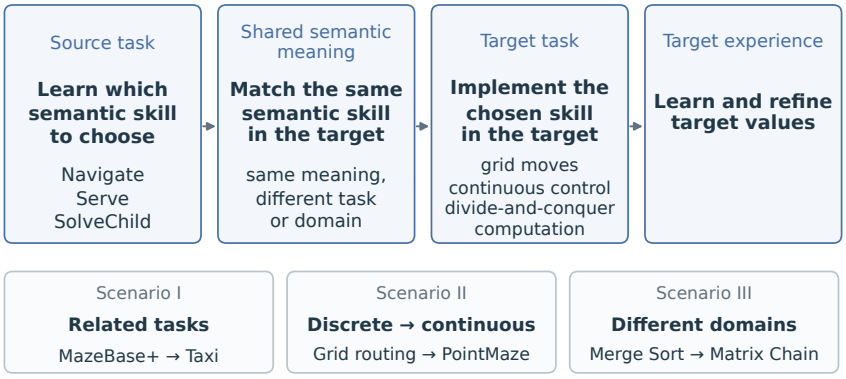


Figure 1: **How semantic-skill transfer works.** A learned source policy selects a semantic skill. The same meaning is matched in the target, which implements the skill in its own way and learns target values from experience. We study this across related tasks, discrete-to-continuous control, and divide-and-conquer domains.

- **Transfer from discrete to continuous control.** Hierarchical knowledge is carried from discrete state and action spaces to continuous states and continuous control by replacing lower-level realizations while preserving higher-level semantic structure. Our main example transfers a learned discrete routing policy to continuous PointMaze.
- **Transfer across apparently different domains.** Learned hierarchical relationships are adapted to apparently different domains by separating reusable semantic structure from target-specific choices, including recursive decompositions. In particular, we transfer a divide-and-conquer structure learned from Merge Sort to Matrix Chain: the learned policy determines how subproblems are solved and combined, while the target problem determines how those subproblems are formed.

Across these settings, transfer reduces new target learning. In Taxi, transferred skills yield useful early behavior and can then be refined. In PointMaze, the transferred high-level policy works with continuous control; on 40 new layouts, transfer improves early return on all 40 and cuts target-learning interactions on 35. In Matrix Chain, Merge Sort's divide-and-conquer structure transfers and yields immediate competence in recursive execution.

Semantic skills also give a natural interface for agentic systems: agents can reason and communicate via subtasks such as `Navigate`, `Search`, `Serve`, or `Combine`, even when implementations differ. This aids interpretation and eases communication, coordination, and cooperation. Related ideas appear in systems that reason over named tools or skills (Yao et al., 2023; Schick et al., 2023; Ahn et al., 2022). Here we take the high-level skill correspondence as given and study transfer of a learned policy over these shared skills across tasks or domains.

Related work has studied reusable skills and policy composition, goal-conditioned policies, successor-feature transfer, and abstractions across related decision processes (Konidaris et al., 2012; Andreas et al., 2017; Schaul et al., 2015; Barreto et al., 2017; Ravindran & Barto, 2003; Ferns et al., 2004). Our focus is complementary: we transfer a learned policy over explicitly matched semantic skills when the target task or the implementation of those skills changes; a more complete comparison appears in App. A.

2 How semantic skills transfer

The previous section stated the idea informally: a learned high-level decision can be reused when its meaning is preserved, even if its implementation changes. We now make this precise; Fig. 1 summarizes the transfer process.

From learned policies to semantic skills. In a hierarchy, one policy chooses among semantic skills such as `Navigate` or `Serve`. Each skill is itself another policy that carries out that choice using lower-level actions. For example, when the higher-level policy chooses `Navigate`, a grid-world skill may move through a sequence of discrete states, while a continuous-control skill may use a

learned controller. The higher-level policy only needs to know which skill to choose and what happens when that skill finishes.

Let s be the current state and let u be a semantic skill. Executing u produces a cumulative discounted reward G and ends in a new state s'. We write Γ for the discount applied when the skill finishes. At the higher level, this gives a transition (s, u, G, Γ, s'). This is the standard semi-Markov view of temporally extended actions (Sutton et al., 1999). The same construction can be repeated across several levels, so that a learned policy at one level becomes a possible choice at the level above.

Connecting the source and target through semantic meaning. Suppose a source task S and a target task T share the same high-level meanings. Their detailed states may be different, so we describe both using the semantic information needed by the higher-level policy. Let

$$e_S : \mathcal{S}_S \to \mathcal{Z}, \qquad e_T : \mathcal{S}_T \to \mathcal{Z}$$

map source and target states to a common semantic description $z \in \mathcal{Z}$. For example, z may record whether an object has been retrieved or whether a subproblem has been solved.

The source task has learned a policy $\bar{\pi}_S(u_S \mid z)$, which gives the probability of choosing source skill u_S in semantic state z. We also know which target skill has the same meaning. Let $m_s : \mathcal{U}_S \to \mathcal{U}_T(s)$ describe this correspondence at target state s. The transferred target policy is

$$b_T(u_T \mid s) = \sum_{u_S :\, m_s(u_S) = u_T} \bar{\pi}_S(u_S \mid e_T(s)). \tag{1}$$

When the correspondence is one-to-one, this simply relabels the source policy. (1) therefore transfers *which semantic skill to choose*, not how that skill is implemented.

The target implements the skill in its own way. After the transferred policy chooses a semantic skill u, the target uses its own controller to carry out that skill. We write $\rho_T(a \mid s, u)$ for this target controller, where a is an environment action. Target execution therefore has two steps:

$$u \sim b_T(\cdot \mid s), \qquad a_t \sim \rho_T(\cdot \mid s_t, u).$$

This separation allows transfer across different action spaces. For example, the source policy may choose `NavigateEast`. In a grid world, this is a discrete move. In PointMaze, the same high-level choice is implemented by a continuous controller that moves toward a neighboring region. The source and target therefore do not need to share low-level actions. They only need to share the semantic structure used by the transferred policy.

Transfer the policy, learn target values. We transfer the source policy over semantic skills, but we learn the target values from target experience. This is important because rewards, skill durations, and low-level dynamics can change from the source task to the target task.

After a semantic skill u finishes, let x' be the next higher-level state. Let x denote the information used by the higher-level value function at a decision point, and let Q^- denote the value estimate used to form the learning target. A standard higher-level Q-learning target is

$$Y = G + \Gamma \max_{v \in \mathcal{U}(x')} Q^-(x', v),$$

with the second term set to 0 when the task ends. With η the learning rate, the target value is updated:

$$Q_T(x, u) \leftarrow Q_T(x, u) + \eta \big[Y - Q_T(x, u) \big] .$$

At the beginning of target learning, the transferred policy provides the high-level decisions. As target experience is collected, the target value function can refine those decisions.

The rest of the paper studies these three forms of transfer in turn. In Scenario I, semantic skills are reused across related tasks. In Scenario II, the same semantic decisions are carried from discrete state and action spaces to continuous states and continuous control. In Scenario III, a divide-and-conquer relationship learned in one computational domain is adapted to another.

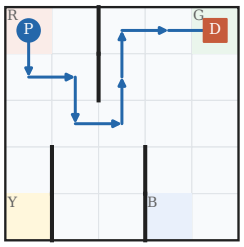
Acquired roles, supplied correspondence

Source choice	p	Taxi role
Reach key	0.731 →	Reach passenger
Pick up key	1.000 →	Pick up passenger
Reach door	0.960 →	Reach destination
Open door	1.000 →	Drop off passenger

p: source probability at the matching semantic state

Recorded here: pickup → navigate → dropoff

Saved Taxi execution



10 native actions; return 11

Figure 2: **Transferring the learned sequence to Taxi.** The source and target use corresponding semantic skills. The transferred policy chooses the Taxi skills, while a Taxi-specific navigator implements `NavP` and `NavD`. The example on the right shows a successful transferred trajectory from passenger pickup to final delivery.

3 Scenario I: Transferring Semantic Skills Across Related Tasks

We first consider the simplest form of transfer: the source and target tasks are different, but they require a similar sequence of high-level decisions.

MazeBase+ and Taxi provide an example. In MazeBase+, the agent learns to navigate to a required object, retrieve it, navigate to a destination, and complete the task. Taxi has the same high-level structure: navigate to the passenger, pick up the passenger, navigate to the destination, and drop off the passenger. The environments and low-level actions are different, but the order of the semantic decisions is closely related. Detailed task and reward definitions are given in App. C.2.1.

From MazeBase+ to Taxi. We transfer the learned source policy by matching its semantic skills to the corresponding Taxi skills. For a Taxi state s, we use the semantic description

$$e_{\text{Taxi}}(s) = (b_P, b_D, b_C, b_F),$$

where b_P describes proximity to the passenger before pickup, b_D describes proximity to the destination while carrying the passenger, b_C records whether the passenger has been picked up, and b_F records whether the delivery has been completed. The corresponding Taxi skills are

$$\mathcal{U} = \{\texttt{NavP}, \texttt{Pickup}, \texttt{NavD}, \texttt{Dropoff}, \textsc{End}\}.$$

Here `NavP` means navigating to the passenger, and `NavD` means navigating to the destination (Fig. 2). The learned source policy determines which semantic skill to choose. A goal-conditioned Taxi navigator then implements the two navigation skills, using the passenger location or destination as its goal.

The semantic description tells the transferred policy which part of the task it is solving, but it does not provide the low-level route. The Taxi navigator still learns how to reach each goal in the target environment. Before any high-level Taxi learning, the transferred policy succeeds from all 300 official initial states. Thus the source policy already provides a successful high-level strategy before the target has learned its own high-level values.

Refining the transferred policy. The transferred policy already gives useful behavior, but the target Q-values are learned from scratch. This creates a problem during early fine-tuning: after a transferred action is tried, its estimated value may become < 0, while an untried action can still keep its initial value of 0. Ordinary greedy selection would then prefer the untried action, even without evidence that it is better. Ties can cause the same problem. We therefore keep the transferred action as the default and compare it only with alternatives that have actually been tried. Let the action preferred by the transferred policy be $b(s) = \arg\max_u b_T(u \mid s)$ and let $N(s, u)$ count how often target experience has updated action u at state s. During target learning, the greedy choice is restricted to $\mathcal{A}_N(s) = \{b(s)\} \cup \{u : N(s, u) > 0\}$. The corresponding Q-learning target is

$$Y = G + \Gamma \max_{v \in \mathcal{A}_{N^-}(s')} Q^-(s', v).$$

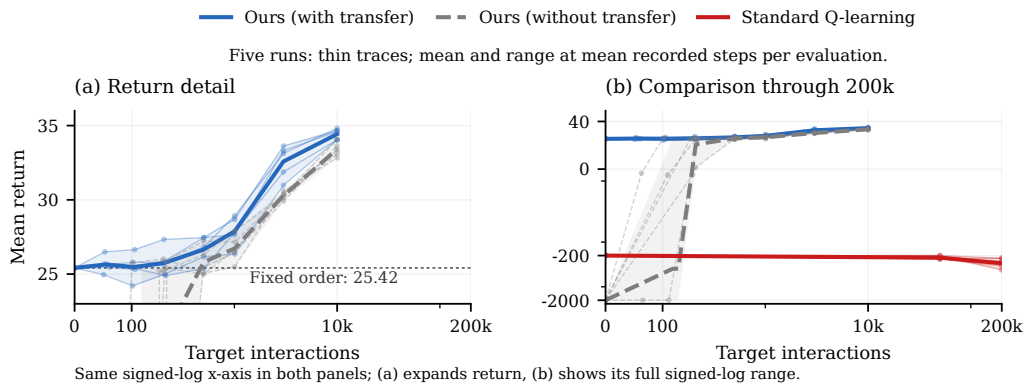


Figure 3: **Reuse the learned skill and learn the request order.** Across 5 runs, after 10,000 target steps the transfer method reaches mean return 34.04–34.81, compared with 32.83–34.06 for the same hierarchy without transfer, and has a larger area under the return curve over the learning window in all 5 runs. Standard Q-learning learns much more slowly; its recorded return remains below the corresponding transfer performance at 500,000 training steps in all five runs. Both panels use the same signed-log interaction axis; the left expands the return range, while the right shows the full range on a signed-log return axis.

An untried action therefore cannot replace the transferred choice merely through initialization. Once tried, its learned value competes with the transferred action as usual. We compare this method with the same hierarchy without transfer: both use the same Taxi semantic description and learned navigator, while transfer also uses the learned source policy and the refinement rule above. Across five runs, transfer maintains success on all 300 initial states during fine-tuning; over the first 20,000 target steps, its normalized success area exceeds the no-transfer hierarchy by 0.247–0.267 in every run. Learning curves and acquisition costs are in App. C.2.1.

Reusing a learned skill while learning a new decision. The previous experiment transfers the complete high-level sequence. We next reuse part of this learned behavior while the target learns a new decision. We treat the learned Taxi behavior as a skill `Serve`(p), where p denotes a passenger request. The skill navigates to passenger p, picks up the passenger, navigates to the requested destination, and completes the delivery. The target problem defined in App. C.2.2 contains three requests. The transferred `Serve` skill completes one request, while a new target policy learns which request to serve next. Thus the target does not need to relearn how to serve a passenger; it only needs to learn the request order.

The transferred `Serve` skill is available before the target scheduler is trained, so target learning can focus on the new scheduling decision. Fig. 3 shows the main benefit of transfer: the learned `Serve` skill can be used immediately, leaving the target to learn only the new request-ordering decision.

Together, these two Taxi experiments illustrate the first form of transfer. Semantic skills learned in one task can provide useful behavior in a related task, and target learning can either refine those skills or use them as building blocks for a new higher-level decision.

4 Scenario II: Transferring from Discrete to Continuous Control

We consider a larger change between source and target. The source uses discrete states and actions, while the target has continuous states and continuous control. The key idea is unchanged: the high-level semantic decision is transferred, while the target implements that decision in its own way.

PointMaze provides our main example in this section. In the discrete source problem, a routing policy chooses a direction such as north, south, east, or west. In the continuous target problem, the same decision specifies which neighboring region to reach, and a continuous controller learns how to move there. Thus the source transfers *where to go next*, while the target determines *how to get there*. The PointMaze task and reward definition are given in App. C.3.1.

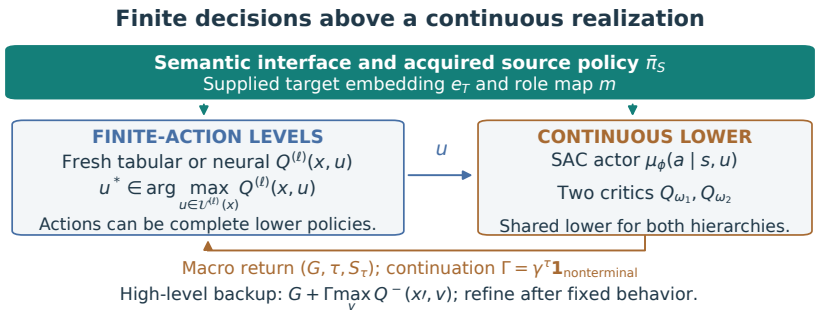


Figure 4: **The same semantic decision can be implemented with continuous control.** The high-level policy chooses a semantic skill such as a routing direction. In the target task, a continuous controller implements that choice. The target learns its own values and controller, while the transferred source policy provides the initial high-level routing decisions.

High-level decisions with continuous states. The set of semantic skills can remain finite even when the target state is continuous. At a high level, the target still chooses among skills such as `North`, `South`, `East`, and `West`. The difference is that the value of each choice now depends on a continuous state. For continuous high-level states, we use a function $Q_\theta(x, u)$, where x is the information used for the high-level decision and u is a semantic skill. The known-maze experiment below keeps a tabular high-level value function, while the unseen-layout experiment uses Q_θ. The learning target has the same form as in Sec. 2. The target learns Q_θ from experience, while the transferred source policy provides the initial high-level decisions.

Implementing a semantic skill with continuous control. A selected semantic skill must then be implemented in the continuous environment. For example, if the high-level policy chooses `East`, the target controller must move the agent toward the neighboring region to the east.

We use a skill-conditioned continuous controller $\mu_\phi(a \mid s, u)$, where u is the selected semantic skill and a is a continuous environment action. In our experiments this controller is learned with soft actor–critic (SAC) (Haarnoja et al., 2018a;b), together with hindsight experience replay (HER) (Andrychowicz et al., 2017). The important point is that the continuous controller is not transferred from the discrete source. The source transfers the high-level routing decision; the target learns how to implement that decision in continuous space (Fig. 4).

From discrete routing to continuous PointMaze. We first use a known seven-region U-shaped maze. The source learns a routing policy on a discrete version of the maze. Its high-level actions are the four cardinal directions. The continuous PointMaze target has the same maze structure, but the agent moves in continuous space. Each high-level direction is matched to a neighboring region, and the target controller moves the agent toward that region. Both the transfer method and the hierarchy without transfer use the same maze information and the same learned continuous controller. The difference is that the transfer method begins with the learned source routing policy.

The transferred policy succeeds on all 84 start–goal cases before any target routing learning. It continues to succeed during target learning. Across five runs, the transfer method reaches the success criterion after about 10,000 additional target interactions. Including the 25,200 interactions used to learn the continuous controller, this corresponds to about 35,200 interactions. A fresh Standard SAC/HER learner reaches the same criterion after 45,000–60,000 interactions, while the same hierarchy without transfer does not reach it within 50,000 target interactions.

These results show that a routing policy learned in a discrete problem can remain useful when the same routing decisions are implemented by continuous control. Additional experimental details and complete learning curves are given in App. C.3.1.

Transfer to previously unseen maze structures. The known-maze experiment keeps the maze structure fixed. We next ask whether the learned high-level routing policy can also help on maze structures that were not present during source learning. For this experiment, the source learns from a collection of discrete training layouts. Its decision uses the relative location of the goal and the available neighboring directions. The target then presents new continuous PointMaze layouts. The

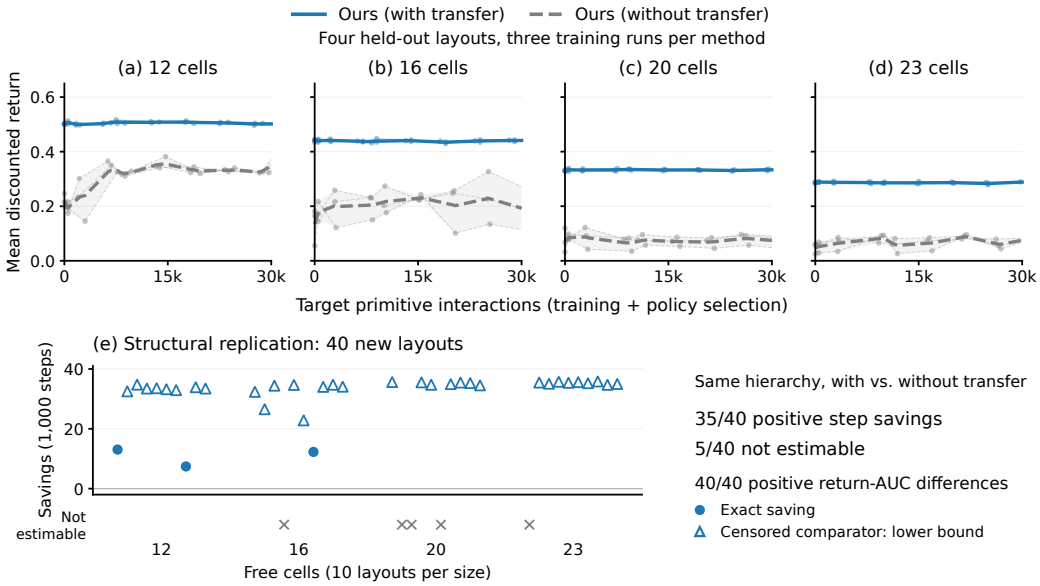


Figure 5: **Transfer reduces learning on unseen PointMaze structures.** The transferred routing policy starts with useful behavior on new layouts and is then refined with target experience. After fixing the procedure, transfer reduces the number of target interactions needed to reach the success criterion on 35 of 40 new layouts, with the median reduction at least 34,246 interactions, and improves the area under the return curve over the first 30,000 interactions on all 40. Dividing the return-area difference by the 30,000-interaction window, the median improvement is 0.211.

transfer method uses the learned source policy to choose high-level directions, while the hierarchy without transfer starts without this learned routing policy. Both methods use the same target information and the same continuous controller.

We first confirmed the transfer procedure on four new layouts with different numbers of regions. We then fixed the complete procedure and evaluated it on 40 additional maze structures, with ten layouts at each of four sizes. Fig. 5 summarizes the main results. Complete layout-level results and the comparison with Standard SAC/HER are in App. C.3.3.

Together, these experiments illustrate the second form of transfer. A high-level policy learned with discrete states and actions can guide learning in a continuous-control problem when the semantic meaning of its decisions is preserved and the target learns how to implement those decisions.

5 Scenario III: Transferring Hierarchical Relationships

We finally consider transfer between problems that appear very different. Merge Sort recursively divides an array into smaller parts and merges the sorted results. Matrix Chain asks how a sequence of matrix multiplications should be decomposed, because different decompositions can have different computational costs. Despite these differences, both can be organized around the same divide-and-conquer structure: split a problem into two smaller subproblems, solve the two subproblems, and combine their solutions. Detailed task and reward definitions are given in Apps. C.4.1 and C.4.2.

This shared structure separates two decisions. The first is *how to execute a given decomposition*: which subproblem to solve next, when to combine the two results, and when the current problem is finished. The second is *how to choose the decomposition itself*. We transfer the first decision from Merge Sort to Matrix Chain, while leaving the decomposition choice to the target problem.

A learned divide-and-conquer policy. For a problem that has been divided into two subproblems, let $z = (b_0, b_1, b_2)$ record whether the current problem, the first subproblem, and the second subproblem have been solved. The source policy learned from Merge Sort chooses among four semantic skills: `SolveChild1`, `SolveChild2`, `Combine`, END. For the states used by the policy,

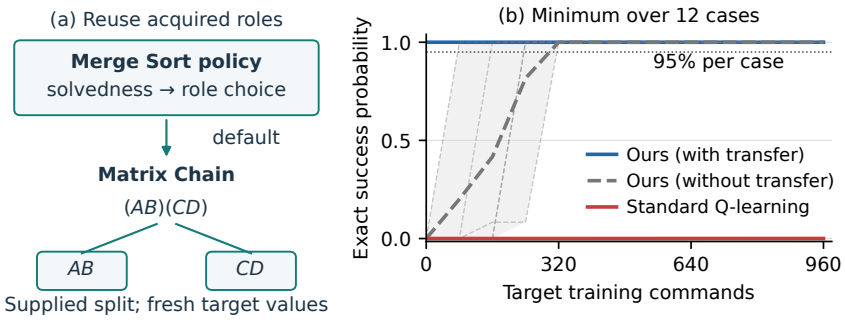


Figure 6: **Transfer the divide-and-conquer structure from Merge Sort to Matrix Chain.** Across five runs, the transferred policy succeeds on all twelve evaluation problems before any target learning and remains successful throughout training. Without transfer, the first evaluation at which every problem reaches at least 95% success occurs after 80–320 target decisions, with a median of 240. Standard Q-learning does not reach this criterion within 960 target decisions.

the learned decision is

$$\bar{\pi}_{\text{recur}}(z) = \begin{cases} \textsc{End}, & b_0 = 1, \\ \texttt{SolveChild2}, & b_0 = 0,\ b_2 = 0, \\ \texttt{SolveChild1}, & b_0 = 0,\ b_2 = 1,\ b_1 = 0, \\ \texttt{Combine}, & b_0 = 0,\ b_1 = b_2 = 1. \end{cases}$$

This policy describes a divide-and-conquer relationship: solve both subproblems, combine their solutions, and then finish. The particular learned policy solves the second subproblem first, but either order is valid. The same semantic decisions can be used in Matrix Chain. Once a Matrix Chain problem has been divided into two smaller matrix-chain problems, the transferred policy determines when to solve each subproblem and when to combine their results. The target implements these skills using its own matrix operations.

Transferring execution from Merge Sort to Matrix Chain. We first test whether the learned divide-and-conquer policy can be reused when the decomposition is already given. This separates the transfer of the hierarchical relationship from the different problem of choosing where to split.

Both the transfer method and the hierarchy without transfer receive the same information about which subproblems have been solved and use the same Matrix Chain operations. The transfer method begins with the learned Merge Sort policy, while the hierarchy without transfer learns the same high-level decisions from target experience. We count each executed high-level choice among `SolveChild1`, `SolveChild2`, `Combine`, and END as one target decision. The results show that the learned hierarchical relationship transfers even though the source and target are different computational problems. The target does not need to relearn the basic divide-and-conquer procedure before it can solve the new problems. Fig. 6 shows the learning curves, while App. C.4.2 gives complete experimental details and final outcomes.

Separating execution from decomposition. Matrix Chain also requires a target-specific decision: where should each matrix chain be divided? For a product $A_iA_{i+1}\cdots A_j$, a split $k \in \{i, \ldots, j-1\}$ produces the two subproblems $A_i \cdots A_k$ and $A_{k+1} \cdots A_j$. Different splits can have different computational costs. The transferred divide-and-conquer policy does not choose this split. Instead, each possible split defines two subproblems, and the transferred policy determines how those subproblems are solved and combined. A separate target policy can therefore learn which split to choose, while reusing the same transferred policy to implement the resulting decomposition.

Merge Sort is a simpler special case because its decomposition uses a fixed midpoint. Matrix Chain has several possible splits, so the target must make an additional structural choice. This separation allows the learned hierarchical relationship to be reused without assuming that the best decomposition is the same in the source and target problems.

Together, these experiments illustrate the third scenario of transfer: a hierarchical relationship learned in one computational domain is adapted to a different domain by reusing the shared divide-and-conquer structure while leaving target-specific decomposition choices to the target problem.

6 When does a shared semantic decision remain useful?

The three transfer scenarios rely on the same idea: a decision learned in the source should remain useful in the target when its effect is preserved, even if the task or its implementation changes. This raises two natural questions. First, when target experience becomes available, when should it replace the transferred decision? Second, when does changing the implementation of a semantic skill preserve enough of its effect for the transferred policy to remain useful? We study these questions in two steps. We first consider *refinement*: how target learning can improve on a transferred policy without discarding it before evidence shows another decision is better. We then consider *transfer*: how differences between the source and target affect the value of the transferred semantic policy. The resulting theory connects directly to the three scenarios above. App. B gives the full assumptions, statements, and proofs.

Refining a transferred decision. Scenario I begins with a transferred decision that is already useful. Target learning should be able to replace that decision, but only after target experience gives evidence for an alternative. The Taxi refinement rule in Sec. 3 is a version of this idea. Let $b(x)$ be the decision preferred by the transferred policy, and let $g_{Q,N}(x)$ be the greedy decision under the refinement rule. Then

$$g_{Q,N}(x) \neq b(x) \quad \Longrightarrow \quad N(x, g_{Q,N}(x)) > 0, \qquad Q(x, g_{Q,N}(x)) > Q(x, b(x)).$$

Thus the refinement rule changes the transferred decision only to an action that has actually been tried and currently has a larger learned value; an untried action or a tie cannot replace the transferred choice. App. B gives a stronger result: when target data provide valid bounds on action values, a transferred decision can be replaced only when the data certify an improvement.

When does transfer survive a change in implementation? We next ask when a source policy remains useful after the target changes how its semantic skills are implemented.

For a state x and semantic skill u, let $r(x, u) = \mathbb{E}[G]$ be the expected discounted reward collected while the skill is implemented. Let $K(\cdot \mid x, u)$ describe where the skill finishes, including the discount applied to the next high-level decision. These two quantities summarize the effect of choosing u: the reward received and what state is reached next.

We compare three models. The source model is denoted by S. An intermediate model I represents the target after matching the source and target semantic skills, before accounting for differences in how the target implements them. The actual target is denoted by T. This separates two possible sources of error. The difference from S to I measures how the same semantic skill changes between the source and target task. The difference from I to T measures the additional change caused by the target implementation of that skill. Let $(e_r^{\mathrm{sem}}, e_K^{\mathrm{sem}})$ bound the differences in expected reward and next-state behavior from S to I, and let $(e_r^{\mathrm{real}}, e_K^{\mathrm{real}})$ bound the corresponding differences from I to T. Assume rewards have magnitude at most B, and the contribution of future decisions is discounted by at most $\beta < 1$. Define

$$\Delta(e_r, e_K) = \frac{e_r}{1-\beta} + \frac{B e_K}{(1-\beta)^2}.$$

The following result shows how these differences affect transfer.

Theorem 1 (Transfer under changes in semantic effects and implementation)**.** *Under the assumptions in App. B, suppose the transferred policy π is uniformly at most ϵ_S worse than the best source policy using the available skills. Assume that the available skills approximate the target skills within distance h, and that reward and next-state effects change at rates at most L_r and L_K with this distance. Then*

$$0 \leq V_T^* - V_T^\pi \leq \epsilon_S + 2\Delta(e_r^{\mathrm{sem}}, e_K^{\mathrm{sem}}) + 2\Delta(e_r^{\mathrm{real}}, e_K^{\mathrm{real}}) + \Delta(L_r h, L_K h).$$

The bound separates four sources of error: source-policy error, changes in corresponding semantic effects, changes caused by target implementation, and incomplete coverage of target skills. Thus source and target need not share states or low-level actions, provided corresponding semantic decisions have similar effects. App. B gives the full assumptions and proof, and a finite-horizon result.

Together, these results formalize the paper's central principle: transfer the semantic decision while allowing the target to implement and refine it in its own way. The three scenarios above show this principle across related tasks, continuous control, and different computational domains.

Reproducibility statement

We provide complete proofs of the theoretical results and additional technical details in the appendix. The experimental setup, implementation details, hyperparameters, and additional numerical results are also documented in the appendix.

AI use statement

Generative AI tools were used to assist with software implementation and debugging, as well as with drafting and editing portions of the manuscript for clarity and presentation. The core research questions, methodology, theoretical results, and scientific conclusions were developed by the authors. All AI-assisted code and text were reviewed and verified by the authors, who take full responsibility for the final content of the paper.

References

David Abel, D. Ellis Hershkowitz, and Michael L. Littman. Near Optimal Behavior via Approximate State Abstraction. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2915–2923. PMLR, 2016. URL https://proceedings.mlr.press/v48/abel16.html.

David Abel, Nathan Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael L. Littman. Value Preserving State-Action Abstractions. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 1639–1650. PMLR, 2020. URL https://proceedings.mlr.press/v108/abel20a.html.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances, 2022. URL https://arxiv.org/abs/2204.01691v2. Version 2, revised 16 August 2022.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular Multitask Reinforcement Learning with Policy Sketches, 2017. URL https://proceedings.mlr.press/v70/andreas17a.html. ICML, PMLR 70:166–175.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The Option-Critic Architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. doi: 10.1609/aaai.v31i1.10916. URL https://ojs.aaai.org/index.php/AAAI/article/view/10916.

André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, Hado P. van Hasselt, and David Silver. Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/350db081a661525235354dd3e19b8c05-Abstract.html.

Jonathon Cai, Richard Shin, and Dawn Song. Making Neural Programming Architectures Generalize via Recursion. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1704.06611v1.

Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning Action Representations for Reinforcement Learning, 2019. URL https://proceedings.mlr.press/v97/chandak19a.html. ICML, PMLR 97:941–950.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning, 2018. URL https://arxiv.org/abs/1810.08272. arXiv:1810.08272; preprint record used.

Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous Control with Action Quantization from Demonstrations, 2022. URL https://proceedings.mlr.press/v162/dadashi22a.html. ICML, PMLR 162:4537–4557.

Peter Dayan and Geoffrey E. Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 5, 1992. URL https://proceedings.neurips.cc/paper/1992/hash/d14220ee66aeec73c49038385428ec4c-Abstract.html.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer. In *IEEE International Conference on Robotics and Automation*, 2017. URL https://arxiv.org/abs/1609.07088. ICRA 2017; arXiv:1609.07088.

Thomas G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition, 1999. URL https://arxiv.org/abs/cs/9905014. arXiv:cs/9905014; preprint record used.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces, 2015. URL https://arxiv.org/abs/1512.07679v2. Preprint first posted 2015; version 2 revised 4 April 2016.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1802.06070.

Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for Finite Markov Decision Processes. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pp. 162–169, 2004. URL https://arxiv.org/abs/1207.4114. UAI 2004 paper; arXiv deposit 2012.

Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation Metrics for Continuous Markov Decision Processes, 2011. URL https://doi.org/10.1137/10080484X. SIAM Journal on Computing 40(6):1662–1714.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An Open-Source Generalist Robot Policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024. doi: 10.15607/RSS.2024.XX.090. URL https://www.roboticsproceedings.org/rss20/p090.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, 2018a. URL https://proceedings.mlr.press/v80/haarnoja18b.html. ICML, PMLR 80:1861–1870.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications, 2018b. URL https://arxiv.org/abs/1812.05905v2. Version 2, revised 29 January 2019.

Dongge Han and Sebastian Tschiatschek. Option Transfer and SMDP Abstraction with Successor Features, 2022. URL https://www.ijcai.org/proceedings/2022/421. IJCAI 2022:3036–3042, DOI 10.24963/ijcai.2022/421.

Matthew Hausknecht and Peter Stone. Deep Reinforcement Learning in Parameterized Action Space. In *International Conference on Learning Representations, Workshop Track*, 2016. URL https://arxiv.org/abs/1511.04143v4. ICLR 2016 workshop-track paper; arXiv:1511.04143v4.

Michael Kearns and Satinder Singh. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49:209–232, 2002. doi: 10.1023/A:1017984413808. URL https://link.springer.com/article/10.1023/A:1017984413808.

George Konidaris, Ilya Scheidwasser, and Andrew G. Barto. Transfer in Reinforcement Learning via Shared Features, 2012. URL https://jmlr.org/papers/volume13/konidaris12a/konidaris12a.pdf. JMLR 13:1333–1371.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning Multi-Level Hierarchies with Hindsight. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1712.00948v5. ICLR 2019 accepted paper; arXiv:1712.00948v5.

Boyan Li, Hongyao Tang, Yan Zheng, Jianye Hao, Pengyi Li, Zhen Wang, Zhaopeng Meng, and Li Wang. HyAR: Addressing Discrete-Continuous Action Reinforcement Learning via Hybrid Action Representation. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2109.05490v3. Accepted version, arXiv:2109.05490v3.

Chengtao Li, Daniel Tarlow, Alexander L. Gaunt, Marc Brockschmidt, and Nate Kushman. Neural Program Lattices. In *International Conference on Learning Representations*, 2017. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2017/03/NPL-final.pdf.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a Unified Theory of State Abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006. URL https://thomasjwalsh.net/pub/aima06Towards.pdf.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-Efficient Hierarchical Reinforcement Learning, 2018. URL https://arxiv.org/abs/1805.08296. NeurIPS 2018, arXiv:1805.08296.

Open X-Embodiment Collaboration. Open X-Embodiment: Robotic learning datasets and RT-X models, 2023. URL https://arxiv.org/abs/2310.08864. Author-provided 2023 preprint record.

Ronald Parr and Stuart J. Russell. Reinforcement Learning with Hierarchies of Machines. In *Advances in Neural Information Processing Systems*, volume 10, 1997. URL https://proceedings.neurips.cc/paper/1997/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

Thomas Pierrot, Guillaume Ligner, Scott E. Reed, Olivier Sigaud, Nicolas Perrin, Alexandre Laterre, David Kas, Karim Beguir, and Nando de Freitas. Learning Compositional Neural Programs with Recursive Tree Search and Planning. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/95b431e51fc53692913da5263c214162-Abstract.html.

Thomas Pierrot, Nicolas Perrin, Feryal Behbahani, Alexandre Laterre, Olivier Sigaud, Karim Beguir, and Nando de Freitas. Learning Compositional Neural Programs for Continuous Control, 2020. URL https://arxiv.org/abs/2007.13363v2. Preprint first posted 2020; version 2 revised 13 April 2021.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL https://jmlr.org/papers/v22/20-1364.html.

Balaraman Ravindran and Andrew G. Barto. SMDP Homomorphisms: An Algebraic Approach to Abstraction in Semi-Markov Decision Processes, 2003. URL https://www.ijcai.org/Proceedings/03/Papers/145.pdf. IJCAI 2003:1011–1016.

Scott Reed and Nando de Freitas. Neural Programmer-Interpreters. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.06279v4. Published ICLR 2016 version; initial arXiv preprint 2015.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320. PMLR, 2015. URL https://proceedings.mlr.press/v37/schaul15.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2997. URL https://proceedings.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Younggyo Seo, Jafar Uruç, and Stephen James. Continuous Control with Coarse-to-fine Reinforcement Learning. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2866–2894. PMLR, 2025. URL https://proceedings.mlr.press/v270/seo25a.html. CoRL 2024; proceedings published 2025.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-Aware Unsupervised Discovery of Skills. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1907.01657v2. Published ICLR 2020 version.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn. ICLR 2021, arXiv:2010.03768.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, 1999. URL https://doi.org/10.1016/S0004-3702(99)00052-1. Artificial Intelligence 112(1–2):181–211.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3540–3549. PMLR, 2017. URL https://proceedings.mlr.press/v70/vezhnevets17a.html.

Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space, 2018. URL https://arxiv.org/abs/1810.06394. arXiv:1810.06394.

Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural Task Programming: Learning to Generalize Across Hierarchical Tasks. In *IEEE International Conference on Robotics and Automation*, 2018. URL https://arxiv.org/abs/1710.01813. ICRA 2018; arXiv:1710.01813.

Sichen Yang and Mauro Maggioni. Multi-level meta-reinforcement learning with skill-based curriculum, 2026. URL https://arxiv.org/abs/2603.08773. arXiv:2603.08773.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629v3. ICLR 2023 camera-ready version; arXiv:2210.03629v3.

Haoqi Yuan, Bohan Zhou, Yuhui Fu, and Zongqing Lu. Cross-Embodiment Dexterous Grasping with Reinforcement Learning, 2024. URL https://arxiv.org/abs/2410.02479. arXiv:2410.02479.

Jinliang Zheng, Jianxiong Li, Dongxiu Liu, Yinan Zheng, Zhihao Wang, Zhonghong Ou, Yu Liu, Jingjing Liu, Ya-Qin Zhang, and Xianyuan Zhan. Universal Actions for Enhanced Embodied Foundation Models, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Zheng_Universal_Actions_for_Enhanced_Embodied_Foundation_Models_CVPR_2025_paper.html. CVPR 2025:22508–22519.

A Related work

Hierarchical reinforcement learning and reusable skills. Hierarchical RL represents long-horizon behavior through temporally extended actions, subroutines, or learned skills (Sutton et al., 1999; Dietterich, 1999; Parr & Russell, 1997; Dayan & Hinton, 1992; Bacon et al., 2017; Vezhnevets et al., 2017). Several lines of work learn reusable skills or compose previously learned policies, including portable options, policy sketches, successor features, and unsupervised skill discovery (Konidaris et al., 2012; Andreas et al., 2017; Barreto et al., 2017; Eysenbach et al., 2019; Sharma et al., 2020). Our focus is different: we study how a learned policy over semantically meaningful skills can be transferred when the target task or the implementation of those skills changes.

Transfer across state and action representations. Goal-conditioned RL and hierarchical methods can reuse goals or subgoals across tasks and continuous-control problems (Schaul et al., 2015; Levy et al., 2019; Nachum et al., 2018; Andrychowicz et al., 2017). Other approaches learn representations of large, parameterized, hybrid, or heterogeneous action spaces (Chandak et al., 2019; Xiong et al., 2018; Li et al., 2022; Dadashi et al., 2022; Seo et al., 2025). Cross-embodiment and language-conditioned systems similarly seek representations that remain meaningful across different controllers or robots (Shridhar et al., 2021; Yuan et al., 2024; Zheng et al., 2025). Here we keep the high-level semantic decision explicit: the source transfers which skill to choose, while the target learns how to implement that skill.

Abstraction and transfer between MDPs. State abstraction, homomorphisms, bisimulation, and successor-feature methods characterize when behavior or value information can be reused across related decision processes (Ferns et al., 2004; Ravindran & Barto, 2003; Li et al., 2006; Barreto et al., 2017). Our theoretical analysis is related in spirit, but is organized around the semantic skills used by the hierarchy. It separates changes in the effect of corresponding skills from changes caused by their target implementation.

Recursive and agentic structure. Neural program induction and recursive-policy methods learn reusable computational procedures (Reed & de Freitas, 2016; Cai et al., 2017; Pierrot et al., 2019). Tool-using and embodied agents also reason through named skills or actions that can be selected independently of their implementation (Ahn et al., 2022; Yao et al., 2023; Schick et al., 2023). Our third scenario studies a related form of reuse in RL: a divide-and-conquer relationship learned in Merge Sort is transferred to Matrix Chain, while target-specific decomposition choices remain separate. More broadly, semantically named skills provide a natural interface for interpretation, communication, and coordination in agentic systems.

B Theory for refining and transferring semantic skills

This appendix gives the full theoretical results supporting Sec. 6. The analysis addresses three questions: when target experience can justify replacing a transferred decision, when a semantic policy remains useful after its implementation changes, and when a transferred divide-and-conquer policy produces a valid recursive computation.

We first define the common setup used by these results. Sec. B.2 states the main results, and Sec. B.3 gives their proofs.

B.1 Setup and assumptions

B.1.1 Semantic decisions and their effects

We describe source and target tasks at the points where a high-level policy chooses a semantic skill. Let $\mathcal{X}$ denote the complete state space needed to predict high-level values, and let U denote the available semantic skills.

The state $x \in \mathcal{X}$ contains all information needed to predict what happens after a semantic skill is chosen. It may include the environment state together with additional information such as task progress, recursive context, or the remaining episode length. The smaller semantic description $z = e(x)$ may be sufficient for choosing a skill, but it need not contain all information needed to predict its value.

Consider choosing semantic skill $u \in U$ at state x. Let G be the discounted reward collected until that skill finishes, let X' be the state at which the next high-level decision is made, and let D indicate whether the overall task has ended. We summarize the effect of the skill by

$$r(x,u) = \mathbb{E}[G \mid x,u], \qquad K(A \mid x,u) = \mathbb{E}[\gamma^\tau \mathbf{1}\{D=0,\ X' \in A\} \mid x,u],$$

where τ is the number of steps on the discounting clock.

Thus $r(x,u)$ describes the expected reward obtained while implementing the skill, while $K(\cdot \mid x,u)$ describes both where the process continues and how future value is discounted. If the overall task ends, there is no future contribution.

We assume that every semantic skill finishes with probability one and that the expectations above are finite. When a recursive subproblem finishes but the full task continues, control returns to the surrounding problem rather than being treated as global termination.

We assume the standard measurability conditions needed for the expectations, Bellman operators, and skill choices below to be well defined. In particular, the maximization over available skills can be approached by measurable choices.

For a policy π, define

$$r_\pi(x) = \int_U r(x,u)\pi(du \mid x), \qquad K_\pi f(x) = \int_U \int_{\mathcal{X}} f(y) K(dy \mid x,u)\pi(du \mid x).$$

The corresponding policy and optimality operators are

$$T_\pi f = r_\pi + K_\pi f, \qquad Tf(x) = \sup_{u \in U} \{r(x,u) + K_u f(x)\}. \tag{B.1}$$

B.1.2 Comparing source and target tasks

The transfer result compares three models defined using the same state description and semantic skills.

The source model is denoted by S. The intermediate model I represents the target after corresponding source and target skills have been matched, but before accounting for differences in how those skills are implemented in the target. The actual target model is denoted by T.

This separates two ways in which transfer can become inaccurate. First, a semantic skill may have a different effect in the source and target task. Second, the target may implement that skill differently from the behavior represented by the matching step.

For models S and I, assume that

$$|r_S(x,u) - r_I(x,u)| \le e_r^{\text{sem}}, \qquad \|K_S(\cdot \mid x,u) - K_I(\cdot \mid x,u)\|_{\text{var}} \le e_K^{\text{sem}}.$$

For models I and T, assume that

$$|r_I(x,u) - r_T(x,u)| \le e_r^{\text{real}}, \qquad \|K_I(\cdot \mid x,u) - K_T(\cdot \mid x,u)\|_{\text{var}} \le e_K^{\text{real}}.$$

These bounds hold uniformly over the states and semantic skills being compared. The norm is

$$\|\mu\|_{\text{var}} = \sup_{|f| \le 1} \left| \int f \, d\mu \right|.$$

The transferred policy uses a subset of available semantic skills $U_h \subseteq U$. The transferred policy may use a finite set of semantic skills U_h to approximate a larger target set U. Assume that every target skill is within distance h of some skill in U_h. We also assume that the target reward and next-state effects change smoothly with this distance:

$$|r_T(x,u) - r_T(x,v)| \le L_r d(u,v), \qquad \|K_T(\cdot \mid x,u) - K_T(\cdot \mid x,v)\|_{\text{var}} \le L_K d(u,v).$$

Finally, the transferred source policy π is assumed to be uniformly within ϵ_S of the best source policy using U_h. These assumptions separate the main sources of transfer error: the source policy itself, changes in the effects of corresponding skills, changes caused by their target implementation, and incomplete coverage of target skills.

B.1.3 DISCOUNTED AND FINITE-HORIZON SETTINGS

For the discounted results, assume

$$|r(x,u)| \le B, \qquad K(\mathcal{X} \mid x,u) \le \beta < 1$$

for every state and semantic skill in each model. Under these assumptions, the policy and optimality operators in (B.1) are contractions, and their value functions are bounded by $B/(1-\beta)$.

Some of our experiments, including the recursive computations in Scenario III, use undiscounted finite episodes. Those problems are treated separately rather than by setting $\beta = 1$. We include the remaining episode length in the state and assume that every nonterminal high-level decision reduces it. When the remaining length reaches zero, the episode ends and there is no future contribution.

For an episode with maximum length H, the finite-horizon analysis uses the same reward and next-state comparisons at every remaining length. The finite-horizon result is stated in Sec. B.2.2.2 and proved in Sec. B.3.2.3.

B.2 MAIN RESULTS

The results below answer the three questions introduced at the beginning of this appendix. We first study when target experience can justify changing a transferred decision. We then bound the performance loss when corresponding semantic skills have different effects or are implemented differently in the target. Finally, we give conditions under which the transferred divide-and-conquer policy produces a valid recursive computation.

B.2.1 REFINING A TRANSFERRED DECISION

B.2.1.1 When target evidence justifies a replacement. Suppose a transferred policy π is used as the current policy. For each state x and alternative skill u, let $L(x,u)$ be a lower bound on the value of choosing u, and let $U(x)$ be an upper bound on the value of following π. Assume that all of these bounds are simultaneously valid with probability at least $1-\delta$.

If

$$L(x,u) \ge U(x),$$

then the target data provide evidence that choosing u is no worse than following the transferred policy at state x.

Proposition B.1 (Improvement from certified replacement)**.** *In a fixed model satisfying either the discounted or finite-horizon conditions in Sec. B.1.3, let π be a baseline policy with true values V^π and Q^π. Assume there is an event $\mathcal{E}$ with probability at least $1-\delta$ on which, simultaneously for every state x and every skill u that the replacement policy may select,*

$$L(x,u) \le Q^\pi(x,u), \qquad V^\pi(x) \le U(x).$$

At each state, replace π only by a distribution supported on skills satisfying $L(x,u) \ge U(x)$, and retain π wherever no such skill is available. Then, on $\mathcal{E}$, every measurable state-dependent mixture of this replacement and π has value at least V^π.

Prop. B.1 gives a general condition for replacing a transferred decision using target evidence. Its proof is given in Sec. B.3.1.1.

B.2.1.2 Guarantee of the Taxi refinement rule. The Taxi experiment in Scenario I uses a simpler rule. It does not estimate the bounds in Prop. B.1. Instead, it keeps the transferred choice available and compares it only with choices that have already been tried.

Proposition B.2 (Property of the Taxi refinement rule)**.** *Fix finite values $Q(x,u)$, nonnegative counts $N(x,u)$, and an available transferred choice $b(x)$. Define*

$$E_N(x) = \{b(x)\} \cup \{u : N(x,u) > 0\},$$

and let $g_{Q,N}(x)$ maximize $Q(x,u)$ over $E_N(x)$, with ties resolved in favor of $b(x)$. Then

$$g_{Q,N}(x) = b(x) \quad \Longleftrightarrow \quad Q(x,b(x)) = \max_{u \in E_N(x)} Q(x,u).$$

Consequently,

$$g_{Q,N}(x) \ne b(x) \quad \Longrightarrow \quad N(x, g_{Q,N}(x)) > 0, \qquad Q(x, g_{Q,N}(x)) > Q(x,b(x)).$$

Prop. B.2 states the guarantee provided by the refinement rule used in Taxi: an untried choice or an exact tie cannot replace the transferred decision. It does not guarantee that every replacement is truly better, because the learned Q values may still contain estimation error. The proof and a counterexample illustrating this distinction are given in Sec. B.3.1.2.

B.2.2 Transfer when the implementation changes

B.2.2.1 Discounted setting. Under the assumptions in Secs. B.1.1–B.1.3, define

$$\Delta(e_r, e_K) = \frac{e_r}{1-\beta} + \frac{Be_K}{(1-\beta)^2}.$$

The first term measures the accumulated effect of reward differences. The second measures the effect of differences in where a skill finishes and how future value is discounted.

For the source-to-target comparison, $(e_r^{\text{sem}}, e_K^{\text{sem}})$ measures the difference between corresponding semantic skills, while $(e_r^{\text{real}}, e_K^{\text{real}})$ measures the additional difference caused by how the target implements them.

The main transfer theorem provides the following bound.

Theorem 1 (restated). Assume: (i) source, intermediate, and target models use the same Markov state space and the same available transferred skill set U_h; (ii) rewards are bounded by B and discounted next-state effects have total mass at most $\beta < 1$; (iii) the source-to-intermediate and intermediate-to-target reward and next-state errors are uniformly bounded by $(e_r^{\text{sem}}, e_K^{\text{sem}})$ and $(e_r^{\text{real}}, e_K^{\text{real}})$; (iv) every target skill is within distance h of some skill in U_h, and target reward and next-state effects are L_r- and L_K-Lipschitz with this distance; and (v) the transferred policy π is uniformly at most ϵ_S worse than the best source policy using U_h. Then

$$\begin{aligned} 0 \le V_T^* - V_T^\pi \le \epsilon_S &+ 2\Delta(e_r^{\text{sem}}, e_K^{\text{sem}}) \\ &+ 2\Delta(e_r^{\text{real}}, e_K^{\text{real}}) + \Delta(L_r h, L_K h). \end{aligned}$$

The four terms have direct interpretations:

1. ϵ_S measures the error of the source policy;
2. $\Delta(e_r^{\text{sem}}, e_K^{\text{sem}})$ measures changes in the effects of corresponding semantic skills;
3. $\Delta(e_r^{\text{real}}, e_K^{\text{real}})$ measures additional changes caused by the target implementation;
4. $\Delta(L_r h, L_K h)$ measures how well the available semantic skills cover the target choices.

Thm. 1 therefore formalizes the intuition behind Scenarios II and III: the source and target may use different states and low-level actions, but transfer remains useful when corresponding semantic decisions produce sufficiently similar effects.

The supporting lemmas are given in Sec. B.3.2.1, and the proof of Thm. 1 is given in Sec. B.3.2.2.

B.2.2.2 Finite-horizon version. For an episode with maximum length H, define

$$\Delta_H(e_r, e_K) = He_r + \frac{1}{2}BH(H-1)e_K.$$

Theorem B.3 (Finite-horizon transfer)**.** *Assume the common state and skill setup of Secs. B.1.1 and B.1.2. Assume that the remaining episode length $t \in \{0, \dots, H\}$ is included in the state.*

After each high-level decision, the process either terminates or moves to a state with a smaller remaining episode length. When no decisions remain, the future value is zero. Any reward associated with reaching the episode limit is included in the transition that reaches it.

Assume that rewards have magnitude at most B; that the source-to-intermediate and intermediate-to-target reward and next-state differences are uniformly bounded, at every remaining episode length, by $(e_r^{\text{sem}}, e_K^{\text{sem}})$ and $(e_r^{\text{real}}, e_K^{\text{real}})$; that every target skill is within distance h of an available skill and the corresponding target effects change at rates at most L_r and L_K; and that the

transferred source policy is uniformly at most $\epsilon_{S,H}$ worse than the best available source policy at the compared initial horizon. Then

$$0 \le V^*_{T,H} - V^{\pi}_{T,H} \le \epsilon_{S,H} + 2\Delta_H(e_r^{\mathrm{sem}}, e_K^{\mathrm{sem}}) + 2\Delta_H(e_r^{\mathrm{real}}, e_K^{\mathrm{real}}) + \Delta_H(L_r h, L_K h).$$

Thm. B.3 has the same interpretation as the discounted result, but the error accumulates over at most H remaining decisions rather than through an infinite discounted sequence.

The proof of Thm. B.3 is given in Sec. B.3.2.3.

B.2.3 Valid divide-and-conquer implementation

The final result supports Scenario III. Even when the decomposition is given, the transferred divide-and-conquer policy must finish the recursive computation and return the correct result.

Proposition B.4 (Valid divide-and-conquer implementation)**.** *Assume a finite decomposition in which every child is strictly smaller than its parent. Assume that the base cases are correct, that once a subproblem has been solved it remains solved, that the solved-state information remains correct, that each solved child returns its result to the calling problem, and that* `Combine` *correctly combines two solved subproblems. Assume that the transferred policy is queried only at the five semantic states listed below, and that the episode is long enough to complete the recursion. If the transferred policy follows*

$$000 \mapsto \texttt{SolveChild2}, \quad 001 \mapsto \texttt{SolveChild1}, \quad 010 \mapsto \texttt{SolveChild2},$$
$$011 \mapsto \texttt{Combine}, \quad 111 \mapsto \textsc{End},$$

then it terminates and returns the correct result for the root problem.

Prop. B.4 concerns the correct implementation of a given decomposition. It does not imply that the decomposition itself is optimal.

The proof of Prop. B.4 is given in Sec. B.3.3.

B.3 Proofs

This section proves the results stated in Sec. B.2. We follow the same organization: first refinement, then transfer under changes in implementation, and finally divide-and-conquer validity.

B.3.1 Proofs for refinement

B.3.1.1 Proof of Proposition B.1. Let π be the transferred policy, with value V^π and action value

$$Q^\pi(x,u) = r(x,u) + K_u V^\pi(x).$$

Let $\mathcal{E}$ be the event on which all lower and upper bounds in Prop. B.1 are simultaneously valid.

At a state x where at least one alternative satisfies

$$L(x,u) \ge U(x),$$

let $\mu(\cdot \mid x)$ be any distribution supported on such alternatives. If no alternative satisfies this condition, let $\mu(\cdot \mid x) = \pi(\cdot \mid x)$.

For any measurable $\lambda(x) \in [0,1]$, define

$$\pi' = (1-\lambda)\pi + \lambda\mu.$$

On the event $\mathcal{E}$, at states where an alternative is certified, every skill in the support of μ satisfies

$$Q^\pi(x,u) \ge L(x,u) \ge U(x) \ge V^\pi(x).$$

At all other states, $\mu = \pi$, so

$$T_{\pi'} V^\pi(x) - V^\pi(x) = 0.$$

Therefore

$$T_{\pi'}V^{\pi}(x) - V^{\pi}(x) = \lambda(x)\left[\int Q^{\pi}(x,u)\mu(du \mid x) - V^{\pi}(x)\right]$$
$$\geq 0.$$

Let

$$A(x) = T_{\pi'}V^{\pi}(x) - V^{\pi}(x).$$

Since

$$V^{\pi'} = r_{\pi'} + K_{\pi'}V^{\pi'},$$

we have

$$(I - K_{\pi'})(V^{\pi'} - V^{\pi}) = A.$$

Under the contraction assumption in Sec. B.1.3,

$$V^{\pi'} - V^{\pi} = \sum_{j=0}^{\infty} K_{\pi'}^{j} A.$$

Every term on the right is nonnegative because $A \geq 0$ and $K_{\pi'}$ preserves nonnegative functions. Hence

$$V^{\pi'}(x) \geq V^{\pi}(x)$$

for every state x on the event $\mathcal{E}$. In the finite-horizon setting, the same argument uses the finite sum $\sum_{j=0}^{H-1} K_{\pi'}^{j}$, since $K_{\pi'}^{H} = 0$. This proves Prop. B.1.

The same argument also applies when the replacement is chosen from a distribution whose expected advantage over the baseline policy is nonnegative, provided the corresponding bound holds.

B.3.1.2 Proof of Proposition B.2.

Recall

$$E_N(x) = \{b(x)\} \cup \{u : N(x,u) > 0\},$$

and let $g_{Q,N}(x)$ maximize $Q(x,u)$ over $E_N(x)$, with ties resolved in favor of the transferred choice $b(x)$.

If

$$Q(x, b(x)) = \max_{u \in E_N(x)} Q(x,u),$$

then $b(x)$ is a maximizer, and the tie-breaking rule selects it. Thus

$$g_{Q,N}(x) = b(x).$$

Conversely, if

$$g_{Q,N}(x) = b(x),$$

then $b(x)$ must attain the maximum over $E_N(x)$. Therefore

$$g_{Q,N}(x) = b(x) \quad \Longleftrightarrow \quad Q(x, b(x)) = \max_{u \in E_N(x)} Q(x,u).$$

Now suppose

$$g_{Q,N}(x) \neq b(x).$$

Because the selected alternative belongs to $E_N(x)$, its count must satisfy

$$N(x, g_{Q,N}(x)) > 0.$$

Because ties are resolved in favor of $b(x)$, a different greedy choice must also satisfy

$$Q(x, g_{Q,N}(x)) > Q(x, b(x)).$$

This proves Prop. B.2.

When the transferred choice is the greedy choice, ϵ-uniform exploration selects it with probability

$$1 - \epsilon + \frac{\epsilon}{|U|}.$$

The implemented Taxi update uses the same pre-update values Q^- and counts N^- when forming the supported maximum. Thus a count changed by the current update does not make a previously untried skill enter the supported set within that same update.

The proposition concerns the greedy part of the refinement rule. Exploration may still select any available skill.

It also does not imply that a replacement has a larger true value. For example, consider a single decision in which the transferred choice has deterministic reward 0, while an alternative gives reward $+1$ with probability $1/4$ and reward -1 with probability $3/4$. The alternative has expected value $-1/2$. After observing one reward $+1$ with learning rate one, it can nevertheless have learned value $Q = 1$, while the transferred choice has value $Q = 0$. The refinement rule then selects the alternative, even though its true value is smaller. This example shows why Prop. B.2 is an exact property of the implemented rule, not a statistical improvement guarantee.

B.3.2 Proofs for transfer when the implementation changes

B.3.2.1 Supporting lemmas. Suppose two models use the same states and semantic skills, their expected rewards differ by at most e_r, and their next-state effects differ by at most e_K, uniformly over states and skills.

For any value function satisfying

$$\|V\|_\infty \le \frac{B}{1-\beta},$$

the corresponding Bellman operators differ by at most

$$c = e_r + \frac{Be_K}{1-\beta}.$$

Because both operators are β-contractions, their fixed points satisfy

$$\|V_A - V_B\|_\infty \le \frac{c}{1-\beta} = \Delta(e_r, e_K). \tag{B.2}$$

The same argument applies to a fixed policy when that policy is used in both models, and to optimal values when the available skill sets agree.

For skill coverage, choose for every target skill $u \in U$ a skill $v \in U_h$ with $d(u, v) \le h$. By the assumptions in Sec. B.1.2, their target action values differ by at most

$$c_h = L_r h + \frac{BL_K h}{1-\beta}.$$

The contraction argument then gives

$$0 \le V_T^* - V_{T,h}^* \le \frac{c_h}{1-\beta} = \Delta(L_r h, L_K h). \tag{B.3}$$

B.3.2.2 Proof of Theorem 1. Let $V_{S,h}^*$ denote the best source value using U_h, and let $V_{T,h}^*$ denote the target optimum restricted to the same skill set.

Decompose

$$\begin{aligned} V_T^* - V_T^\pi = (V_T^* - V_{T,h}^*) + (V_{T,h}^* - V_{S,h}^*) \\ + (V_{S,h}^* - V_S^\pi) + (V_S^\pi - V_T^\pi). \end{aligned} \tag{B.4}$$

By (B.3),

$$V_T^* - V_{T,h}^* \le \Delta(L_r h, L_K h).$$

Applying (B.2) from S to I and then from I to T gives

$$V_{T,h}^* - V_{S,h}^* \le \Delta(e_r^{\text{sem}}, e_K^{\text{sem}}) + \Delta(e_r^{\text{real}}, e_K^{\text{real}}).$$

By the source-policy assumption,

$$V_{S,h}^* - V_S^\pi \le \epsilon_S.$$

Applying the fixed-policy version of (B.2) also gives

$$V_S^\pi - V_T^\pi \le \Delta(e_r^{\text{sem}}, e_K^{\text{sem}}) + \Delta(e_r^{\text{real}}, e_K^{\text{real}}).$$

Combining the four terms yields

$$\begin{aligned} V_T^* - V_T^\pi \le \epsilon_S &+ 2\Delta(e_r^{\text{sem}}, e_K^{\text{sem}}) \\ &+ 2\Delta(e_r^{\text{real}}, e_K^{\text{real}}) + \Delta(L_r h, L_K h). \end{aligned}$$

The left-hand side is nonnegative because the transferred policy is available in the target policy class. This proves Thm. 1.

B.3.2.3 Proof of Theorem B.3. Let d_t denote an upper bound on the difference between two value functions when at most t high-level decisions remain. At $t = 0$,

$$d_0 = 0.$$

For $t \ge 1$, the immediate reward contributes at most e_r, while the next-state difference contributes at most $B(t-1)e_K$. Therefore

$$d_t \le d_{t-1} + e_r + B(t-1)e_K.$$

Summing gives

$$\begin{aligned} d_H &\le \sum_{t=1}^{H} [e_r + B(t-1)e_K] \\ &= He_r + \frac{1}{2}BH(H-1)e_K = \Delta_H(e_r, e_K). \end{aligned}$$

Applying the same argument to skill coverage, and then using the same four-term decomposition as in (B.4), gives the bound in Thm. B.3. This proves the theorem.

B.3.3 Proof of divide-and-conquer validity

We prove Prop. B.4 by induction on subproblem size.

The base cases are correct by assumption. Now consider a larger unsolved problem. Every recursive child is smaller than its parent, so the induction hypothesis applies whenever the policy chooses `SolveChild1` or `SolveChild2`. The chosen child therefore returns the correct result to the calling problem.

Starting from semantic state 000, the transferred policy first chooses `SolveChild2`. After the second child returns, the state becomes 001, so the policy chooses `SolveChild1`. After both children return, the state is 011, and the policy chooses `Combine`. Correct combination gives state 111, after which the policy chooses END.

If the state begins at 010, the first child is already solved, so the policy chooses `SolveChild2` and then proceeds to 011.

Because child sizes strictly decrease, the recursion cannot continue indefinitely. Because solved-state information remains correct, a solved child is not unnecessarily recomputed. Because each child returns its result to the correct calling problem and `Combine` is correct, the final root result is correct.

Only the final END at the root ends the complete task. Finishing an inner subproblem returns its result to the surrounding problem.

Therefore the transferred policy terminates and returns the correct root result, provided the episode is long enough to complete the recursion. This proves Prop. B.4.

For the Matrix Chain implementation with $n \ge 2$ leaves, a valid implementation uses $n-2$ nonroot child-solving decisions, $n-1$ `Combine` decisions, and $n-1$ END decisions, for a total of

$$3n - 4$$

high-level decisions. Thus an episode length of at least $3n-4$ is sufficient for this implementation.

C EXPERIMENTAL DETAILS FOR THE THREE TRANSFER SCENARIOS

This appendix gives the task definitions, training procedures, evaluation protocols, and additional results for the three transfer scenarios studied in the main text. Sec. C.2 covers transfer across related tasks, Sec. C.3 covers transfer from discrete to continuous control, and Sec. C.4 covers transfer across divide-and-conquer domains.

C.1 HOW THE EXPERIMENTS ARE MEASURED

Each target experiment starts with fresh target value estimates. Transferring a policy does not copy its learned numerical Q-values into the target.

We report source training, lower-level policy training, and new target learning separately when they differ. Learning effort is measured in the natural unit of each experiment: target environment interactions for Taxi and PointMaze, and high-level decisions for the recursive experiments.

Results over repeated experiments include all independent training runs.

C.2 SCENARIO I: TRANSFER ACROSS RELATED TASKS

C.2.1 MAZEBASE+ AND TAXI: TRANSFERRING SEMANTIC SEQUENCING

Problem definition. The source task is a key-and-door problem in MazeBase+. The agent must reach a key, pick it up, move to a door, and open the door. The state is

$$s = (x, y, k, o),$$

where (x, y) is the agent position, $k \in \{0, 1\}$ records whether the key has been collected, and $o \in \{0, 1\}$ records whether the door has been opened. There are 108 states and six environment actions: four movement actions, `Pick`, and `Open`. The complete source-task dynamics, rewards, and training reward shaping are given in Sec. C.2.1.

The high-level source policy chooses among five semantic skills:

$$\{\texttt{NavigateKey}, \texttt{Pickup}, \texttt{NavigateDoor}, \texttt{Open}, \textsc{End}\}.$$

The two navigation skills use a learned goal-conditioned navigation policy. The source therefore learns the semantic sequence "navigate to the required object, acquire it, navigate to the final goal, complete the task."

The target is the standard deterministic Taxi task. A state specifies the taxi location, passenger location, and destination. The environment has 500 states and six actions: four movements, pickup, and dropoff. A successful delivery gives reward $+20$, an illegal pickup or dropoff gives -10, and every other environment transition gives -1. Episodes are limited to 200 environment steps. The complete target-task definition is given in Sec. C.2.1.

The Taxi high-level policy uses the semantic skills

$$\{\texttt{NavigatePassenger}, \texttt{Pickup}, \texttt{NavigateDestination}, \texttt{Dropoff}, \textsc{End}\}.$$

Its semantic description records whether the taxi is near the passenger, whether it is near the destination while carrying the passenger, whether the passenger has been picked up, and whether delivery has been completed.

The source-to-target correspondence is

$$\begin{aligned} \texttt{NavigateKey} &\leftrightarrow \texttt{NavigatePassenger}, \\ \texttt{Pickup} &\leftrightarrow \texttt{Pickup}, \\ \texttt{NavigateDoor} &\leftrightarrow \texttt{NavigateDestination}, \\ \texttt{Open} &\leftrightarrow \texttt{Dropoff}, \\ \textsc{End} &\leftrightarrow \textsc{End}. \end{aligned}$$

The correspondence is fixed before target learning. It only matches the meanings of the skills; it does not tell the target which skill is best in a given state.

Exact MazeBase+ dynamics and training rewards. The state is (x, y, k, o), with position $(x, y) \in \{1, \ldots, 9\} \times \{1, 2, 3\}$, key-possession flag $k \in \{0, 1\}$ and door-open flag $o \in \{0, 1\}$: 108 states. The key is at $(1, 3)$; the door is at $(10, 2)$, just outside the room, and can be opened from the adjacent room cell $(9, 2)$. The six environment actions are north, east, south, west, Pick and Open. A legal movement succeeds with probability 0.9 and otherwise stays in place; an out-of-room movement stays in place. Pick at the key changes k from 0 to 1 with probability 0.9. Open with the key and at Manhattan distance at most one from the door changes o from 0 to 1 with probability 0.9. Unsuccessful or inapplicable interactions leave the state unchanged. States with $o = 1$ are absorbing. The initial set is all 54 states with $o = 0$, including both values of k.

The learned high-level state is

$$z = (\mathbf{1}_{(x,y)=(1,3)},\ k\,\mathbf{1}_{\|(x,y)-(10,2)\|_1 \le 1},\ k,\ o).$$

The five high-level choices are Pick, Open, navigate to the key, navigate to the door, and End. The two navigation choices use the previously learned goal-conditioned navigation skill, specialized to the corresponding coordinate; the action features implement stopping beside the door. Pick and Open implement one environment action. A navigation call returns when its lower policy chooses End, the task ends, or its execution limit is reached. End itself leaves the environment state unchanged and makes no environment call; it returns from a skill and is not evidence of task success.

For a transition from $o = 0$, the environment reward is 10000 if $o' = 1$ and -10 otherwise. Absorbing-state transitions have reward 0. The source was trained with additional reward shaping

$$r_{\text{train}}(s, a, s') = r_{\text{env}}(s, a, s') + 5\big(I_K + I_P + I_D\big),$$

where I_K indicates entering the key cell from another cell, I_P indicates $k = 0 \to k' = 1$, and I_D indicates entering Manhattan distance at most one from the door from farther away. These are transition indicators, not once-per-episode bonuses. Discount is 1. End has an internal training penalty of -10 and discount 1; this is not an environment transition reward. The source-training code limits a training trajectory by $\max(t, n) < 50$, where t counts its top-level choices and n counts its environment actions; nested calls receive the remaining allowance. Reaching a limit without opening the door is not success. Updates use the returned state and the saved continuation discount rather than treating the collection limit as a successful terminal state.

The source is the fixed key-and-door level-2 checkpoint. Its tabular training uses exploration probability 0.15, visit-based step sizes $1/N$, and the four-flag representation to export its categorical skill policy. The saved key-and-door training used 1,000 level-1 and 2,000 level-2 training episodes, respectively 15,255 and 34,313 environment actions. Including its navigation prerequisites, the shared source costs 5,822,401 recorded training actions. Historical evaluation overhead is incomplete and is not assigned zero cost. The Taxi experiment uses this fixed policy; it does not retrain or select a MazeBase+ source on target outcomes. Source success is door opening, while the separate 300-state Taxi evaluation tests passenger delivery under the matched semantic description.

Exact Taxi dynamics. The implementation uses deterministic Gymnasium 1.3.0 Taxi-v4, with both rainy movement and fickle destinations disabled. The state is (r, c, p, d): row and column $r, c \in \{0, \ldots, 4\}$, passenger location $p \in \{0, 1, 2, 3, 4\}$, and destination $d \in \{0, 1, 2, 3\}$. Passenger value 4 means inside the taxi; values 0–3 refer to R, G, Y, B at $(0, 0), (0, 4), (4, 0), (4, 3)$, respectively. The state index is $((5r + c)5 + p)4 + d$. North/south movement is limited by the grid boundary. East/west movement is additionally blocked between columns 1 and 2 in rows 0 and 1, between columns 0 and 1 in rows 3 and 4, and between columns 2 and 3 in rows 3 and 4. A blocked move leaves the position unchanged. The six environment actions, in index order, are south, north, east, west, pickup and dropoff.

A legal pickup at the waiting passenger's location sets $p = 4$. A dropoff while carrying at the destination terminates successfully, sets $p = d$, and gives environment reward $+20$. A dropoff while carrying at another landmark leaves the passenger there and gives -1, without ending the task. An illegal pickup or dropoff leaves the state unchanged and gives -10. Every other environment action gives -1. Episodes truncate after 200 environment actions. The initial/evaluation population contains all $25 \times 4 \times 3 = 300$ states with a waiting passenger and distinct pickup and destination landmarks. Truncation without delivery is failure.

Fig. C.1 shows the matched sequence; Fig. 2 shows a Taxi trajectory.

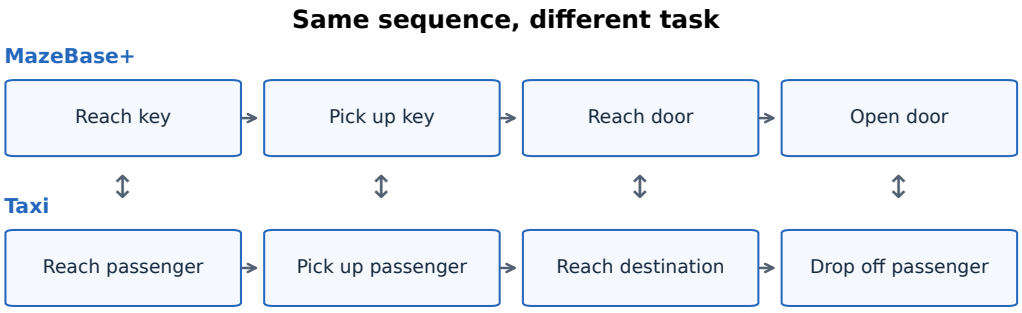


Figure C.1: The learned sequence matches key pickup and door opening to passenger pickup and delivery. Each task implements its own navigation.

Method. The learned MazeBase+ policy gives a categorical distribution over the five semantic skills at each semantic state. During deterministic evaluation, we use its most probable skill.

In Taxi, the two navigation skills use the same learned goal-conditioned navigator, with either the passenger or the destination as the navigation goal. Thus the source policy determines which semantic skill to use, while Taxi determines how that skill is implemented.

The target high-level Q-table and visit counts start from scratch. They use the full 500-state Taxi state, not the smaller semantic description used to query the transferred policy. Thus the transferred policy guides behavior, while target values are learned entirely from Taxi experience.

For target refinement, let $b(s)$ denote the semantic skill preferred by the transferred policy, and let $N(s, u)$ be the number of target updates for skill u at state s. Greedy target improvement considers

$$A_N(s) = \{b(s)\} \cup \{u : N(s, u) > 0\}.$$

The transferred choice is therefore kept as the default until an alternative has been tried, and ties are resolved in favor of the transferred choice. The corresponding theoretical property is given in Prop. B.2.

We compare three methods:

1. the hierarchy with transfer and the refinement rule above;
2. the same hierarchy trained without transfer;
3. standard Q-learning directly on the Taxi environment.

The two hierarchical methods use the same learned Taxi navigator. Two navigators are trained independently for 250,000 environment interactions each and are shared across the five target training runs. The hierarchical target learner uses $\alpha = 0.5$, $\gamma = 0.99$, and $\epsilon = 0.05$. Each target run receives 20,000 additional Taxi interactions.

For the transfer method, the transferred policy guides behavior until the first completed high-level skill at or after 5,000 target interactions. The target Q-values are updated during this period. Afterward, target learning continues using the refinement rule above.

A redundant navigation request receives an internal reward of -1, and a premature END receives an internal reward of -200. These rewards are used only for high-level learning; reported task return uses the original Taxi rewards.

The four source-policy flags are at the waiting passenger, carrying and at the destination, carrying, and delivery completed. The passenger-location flag is zero after pickup. Skill indices are pickup, dropoff, navigation to passenger, navigation to destination, and END. All five remain available for exploration. Without transfer, exact ties use the first action in this order and the learning target maximizes over all five actions. The transfer update uses the values and visit counts from before the update.

The navigator's table has 625 position–goal pairs and four movement actions. It learns with reward zero at its goal and -1 otherwise, step size 0.25, discount 0.99, exploration 0.2, and a 100-action

Method	Measure	201	202	203	204	205
Ours (with transfer)	Success	300	300	300	300	300
	Return	7.837	7.803	7.770	7.857	7.863
Ours (without transfer)	Success	297	299	298	296	298
	Return	−12.573	0.773	0.643	−12.650	0.677
Standard Q-learning	Success	300	300	300	300	300
	Return	7.910	7.917	7.917	7.930	7.917

Table 1: All five final Taxi runs for the three learning methods. Successes are out of 300 and returns are rounded to three decimals. Hierarchies receive 20k additional target steps after a navigator trained for 250k interactions; standard Q-learning receives 270k training interactions. The unchanged source alone has 300/300 success and return 7.930. Complete additional-control tables remain in the experiment record.

episode limit. A target navigation skill stops at its goal, environment termination or truncation, or after 30 actions. Target run identifiers 201, 203 and 205 share one trained navigator; 202 and 204 share the other. The last skill is completed, allowing up to 29 interactions beyond the training budget.

For duration τ, a skill uses $G = \sum_{j<\tau} 0.99^j r_j$ and continuation discount 0.99^τ, with no continuation at successful termination. Redundant navigation uses internal $G = -1$ and discount 0.99 without an environment action. Premature END terminates without an environment action. A navigation skill interrupted by the episode limit before its goal or 30-action return is excluded from bootstrapping. Standard Q-learning uses $\alpha = 0.2$, $\gamma = 0.99$, $\epsilon = 0.05$, and 270,000 training interactions.

Evaluation and additional results. Before any target high-level training, the transferred source policy together with the learned Taxi navigator successfully completes the task from all 300 official initial states.

We then run five independent target-training runs. With transfer, all 300 initial states remain successful at every recorded evaluation during fine-tuning. Over the first 20,000 target interactions, the normalized area under the success curve exceeds that of the same hierarchy without transfer by 0.247–0.267 in every run.

At the final evaluation, the numbers of successful initial states are

with transfer	300	300	300	300	300
without transfer	297	299	298	296	298.

This comparison measures the benefit of reusing a semantic policy that is already available. The MazeBase+ source policy and the Taxi navigator require their own training, so the experiment does not claim that transfer uses less total training than learning Taxi directly.

Hierarchical evaluations occur at $0, 1, 2.5, 5, 7.5, 10, 15, 20$ thousand interactions, including evaluations immediately before and after refinement starts. Standard Q-learning additionally evaluates at $50, 100, 150, 200, 250, 270$ thousand. Evaluation covers every official initial state without learning updates. Success area integrates the piecewise-linear curve against recorded interactions over 0–$20,000$ and divides by 20,000.

Tab. 1 reports all five final comparisons. Figs. 12 and 13 show complete success and return curves. Keeping the source policy fixed also gives success area one and mean return 7.930; refinement is not credited with improving its already complete success. Standard Q-learning reaches 300/300 at the 100,000-interaction evaluation in every run. The shared MazeBase+ source required 5,822,401 known training actions; historical source evaluation overhead is incomplete and is not treated as zero.

C.2.2 MULTI-REQUEST TAXI: REUSING SERVE WHILE LEARNING REQUEST ORDER

Problem definition. The target extends Taxi from one passenger request to three. The taxi can carry at most one passenger at a time, and the task is complete only after all three requests have been

served. The state records the taxi position, the origin and destination of each request, and the current status of each request.

The environment has ten actions: four movement actions and a pickup and dropoff action for each of the three requests. A successful delivery gives reward $+20$, an illegal pickup or dropoff gives -10, and every other environment step gives -1. All three deliveries must be completed within 200 environment steps.

We use the previously learned single-request Taxi behavior as a semantic skill

$$\texttt{Serve}(p),$$

where p denotes one of the three requests. For a selected request, `Serve` navigates to its passenger, picks up the passenger, navigates to the requested destination, and completes the delivery.

The new target decision is therefore the request order. A scheduler chooses which unfinished request to serve next, while `Serve` determines how the selected request is completed.

The grid and movement rules are those in Sec. C.2.1. Request status is 0–3 for waiting at R/G/Y/B, 4 for carried, and 5 for delivered. At most one status is 4. Action indices $4 + 2p$ and $5 + 2p$ are pickup and dropoff for request $p \in \{0, 1, 2\}$. Pickup requires an empty taxi at that waiting passenger; dropoff at another landmark leaves the carried passenger waiting there and gives -1. Other illegal interactions leave the state unchanged. The scheduler and direct Q-learning state contains origins, destinations, position and request statuses, excluding elapsed time; the execution table without transfer uses the four semantic flags. The hierarchy also stops after 200 execution-skill choices.

Fig. C.2 separates the learned request execution from the new scheduling decision.

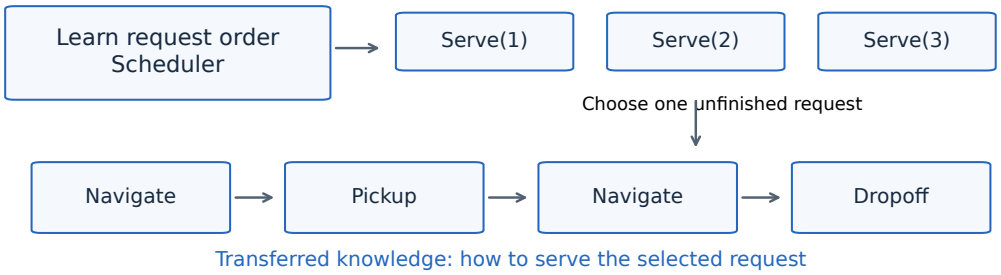


Figure C.2: The scheduler learns which request to select. The transferred `Serve` skill implements the selected request.

Method. We compare three methods:

1. a hierarchy that reuses the learned `Serve` skill and learns only the request scheduler;
2. the same hierarchy without transfer, which learns the execution of each selected request from target experience;
3. standard Q-learning over the ten environment actions.

The two hierarchical methods use the same task state, the same scheduler, and the same learned Taxi navigator. Their schedulers start from fresh Q-values.

For the transfer method, the execution of `Serve` uses the learned single-request Taxi policy. The source policy does not choose the request order. The scheduler learns that decision entirely from the three-request target.

During scheduler training, request choices that have not yet been observed in the current scheduler state are tried before ordinary ϵ-greedy Q-learning is used. The scheduler uses

$$\alpha = 0.5, \qquad \gamma = 0.99, \qquad \epsilon = 0.2.$$

The hierarchy without transfer uses the same learning parameters for its request execution.

Standard Q-learning acts directly through the ten environment actions, with $\alpha = 0.2$ and $\epsilon = 0.2$.

Each hierarchical method receives 10,000 target environment interactions. Standard Q-learning is trained for up to one million target interactions.

Discount is 0.99 per environment action. Completed skills and requests use their discounted environment rewards and continuation 0.99^{τ}. Redundant navigation has internal reward -1 and continuation 1, with no environment action; it affects execution learning only, not scheduler or reported returns. Incomplete skills or requests receive no corresponding higher-level update. Standard Q-learning uses zero continuation at success but retains its one-step bootstrap at the time limit. Reported return is the undiscounted sum of environment rewards.

Execution without transfer is fixed during each selected request, while working values may update at completed skill boundaries. The scheduler samples uniformly among unfinished requests with no completed observation at its current state before using ϵ-greedy choices. Evaluation uses deterministic maximization without updates. Neither distances nor optimal request orders enter learning. The shared navigator required 250,000 training interactions; the shared MazeBase+ policy has the source-training cost stated in Sec. C.2.1.

Evaluation and additional results. We use 48 target configurations fixed before training. They vary the request destinations, initial taxi location, and labels assigned to the three requests. The same 48 configurations are used throughout target training and evaluation, so this experiment studies adaptation to these target configurations rather than generalization to unseen configurations.

Before scheduler training, the transferred `Serve` skill already completes all three requests successfully on all 48 configurations.

We then perform five independent target-training runs. After 10,000 target interactions, the transfer method reaches mean return

$$34.04\text{–}34.81,$$

while the same hierarchy without transfer reaches

$$32.83\text{–}34.06.$$

The transfer method has a larger area under the return curve over the target-learning window in all five runs.

The hierarchy without transfer first reaches success on all 48 configurations after 101–502 target interactions across the five runs. Standard Q-learning learns more slowly: at 500,000 target interactions, its recorded return is still below the corresponding 10,000-step transfer result in every run.

These results show that a learned semantic skill can be reused while the target learns a new higher-level decision. Here, `Serve` handles how to complete one request, while target learning focuses on which request to serve next.

The 48 configurations are the two directed destination cycles for requests originating at R/G/B, all four landmark starting positions, and all six request-label permutations. Evaluations are scheduled at $0, 50, 100, 200, 500, 1,000, 3,000, 10,000$ interactions, at completed-request boundaries and the final hard cutoff. Repeated labels at an unchanged actual count describe one evaluation state. Standard Q-learning additionally has recorded evaluations at $50,000, 200,000, 500,000, 1,000,000$. Evaluation is detached from training.

Fig. 3 shows both hierarchies through 10,000 training interactions and Standard Q-learning through 200,000. The area comparison uses recorded interaction counts over 0–$10,000$, with linear segments between observations. It does not imply higher transfer return at every evaluation: the hierarchy without transfer leads at five intermediate evaluations. Fixed request order has mean return 25.4167; learned scheduling improves on it in all five runs. These results do not claim lower total training cost.

C.3 Scenario II: transfer from discrete to continuous control

C.3.1 PointMaze: discrete-to-continuous transfer on a fixed maze

Problem definition. PointMaze is a continuous-control navigation problem. The agent observes its continuous position and velocity together with a continuous goal, and chooses a two-dimensional action

$$a \in [-1, 1]^2.$$

The agent receives reward 1 when its position is within distance 0.45 of the final goal and 0 otherwise. There is no step penalty or collision penalty, and an evaluation episode ends successfully when the goal is reached. The episode limit is 300 environment interactions.

The fixed maze contains seven connected free regions arranged in a U shape. At the high level, each region is treated as one discrete state. The available semantic decisions are the legal neighboring directions

$$\{\texttt{North}, \texttt{East}, \texttt{South}, \texttt{West}\}.$$

The high-level state records the current region and the region containing the final goal.

The source problem uses this seven-region graph directly. A tabular policy learns which neighboring direction to choose for each ordered pair of current and goal regions. There are 42 distinct ordered start–goal pairs. The source is trained for 5,000 discrete moves and solves all 42 pairs at the end of training. It is learned from reward and experience; shortest-path labels are not used.

The continuous PointMaze target has the same high-level routing decisions, but a chosen direction must now be implemented through continuous control. A high-level direction selects the center of the corresponding neighboring region as an intermediate goal. When the selected neighbor contains the final destination, the controller uses the actual continuous destination instead.

The full environment state contains position p, velocity v, goal g, geometry and elapsed interactions. Observations give (p, v), achieved goal p and desired goal g. Initial positions and goals have independent coordinate-wise uniform offsets in $[-0.25, 0.25]$ from their selected region centers. Actions are clipped to $[-1, 1]^2$ and velocities to $[-5, 5]^2$ before each simulation step. The two sliding joints use damping 1, motor gear 100, timestep 0.01 and frame skip 1; wall collisions follow the simulator dynamics.

The free regions have row/column coordinates $(1, 1), (1, 2), (1, 3), (2, 3), (3, 1), (3, 2), (3, 3)$ within a 5×5 map, with center $(c - 2, 2 - r)$ for region (r, c). All other regions are walls. The high-level table omits velocity, within-region offsets and remaining time; exact state abstraction is not assumed. A navigation skill executes at least one action and stops within radius 0.45 of its intermediate goal, on reaching the final goal, or at the episode limit. Final-goal success takes priority.

Fig. C.3 illustrates the discrete graph and continuous motion. The recorded trajectory is an interface illustration; it is not one of the five target-training comparisons below.

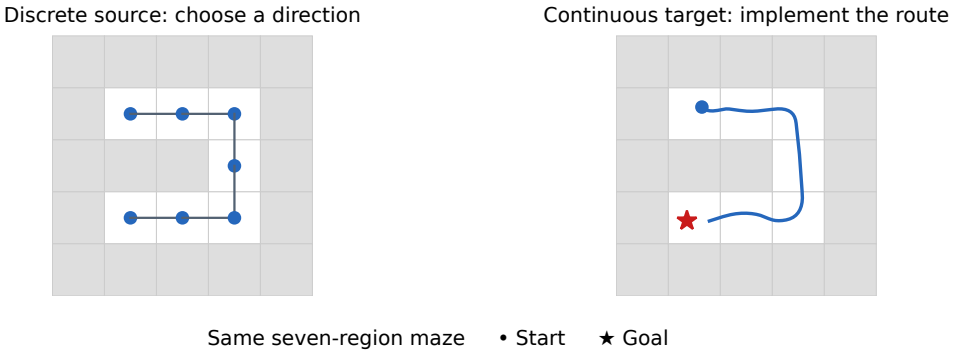


Figure C.3: The seven-region source chooses the next direction; a continuous target controller implements it. The trajectory is a previously recorded successful execution.

Method. We compare three methods:

1. a hierarchy initialized with the learned discrete routing policy;
2. the same hierarchy without transfer;
3. standard SAC with hindsight experience replay (HER), trained directly on the continuous PointMaze task.

The two hierarchical methods use the same maze, the same legal neighboring directions, and the same continuous navigation controller. Only the first method uses the learned discrete routing policy.

For each target run, the continuous navigation controller is trained for 25,200 environment interactions using SAC with HER. It learns to reach neighboring-region goals from continuous states. After this training, the controller is fixed and used by both hierarchical methods in that run.

The target high-level Q-table starts from scratch. When a direction is selected, the continuous controller moves toward the corresponding intermediate goal, and the resulting reward and duration are used to update the high-level value.

For the transfer method, the learned discrete routing policy guides behavior through the first completed high-level decision at or after 10,000 target interactions. Target values are updated during this period. Afterward, behavior is chosen from the target Q-table with exploration probability 0.05.

Without transfer, the target Q-table controls behavior from the beginning. Its exploration probability decreases from 0.30 to 0.05 over the first 10,000 target interactions. Both hierarchical methods receive up to 50,000 new target interactions.

Standard SAC/HER learns the continuous task directly from scratch. It receives the continuous state and final goal, but does not use the intermediate region goals or the high-level routing policy.

The discrete source uses step size 0.2, discount 0.99, exploration decreasing from 0.30 to 0.05 over 4,000 moves, and a 32-decision horizon. Reward is 1 on reaching the goal and 0 otherwise. The final source policy is fixed.

SAC/HER uses two 128-unit hidden layers, learning rate 3×10^{-4}, discount 0.99, batch size 128, replay capacity 100,000, target coefficient 0.005, automatic entropy adjustment, and one gradient update per interaction after 1,000 initial interactions. Future-goal HER uses four relabeled goals per sample (80%). Training continues after reaching a goal until the 300-interaction limit; relabeled goals use the same sparse reward. Replay retains continuation at time limits. Reported episodic returns exclude the SAC entropy term. Standard SAC/HER receives 105,000 training interactions using this same recipe.

For the hierarchical learner, $G = \sum_{j<\tau} 0.99^j r_j$ and $\Gamma = 0.99^\tau$, with $\Gamma = 0$ at episode termination or its time limit. The update is

$$Q(z,u) \leftarrow Q(z,u) + 0.2\{G + \Gamma \max_{v \in \mathcal{U}(z')} Q(z',v) - Q(z,u)\}.$$

Both hierarchies complete their final selected skill. They do not use the Taxi refinement rule. The five independent continuous-controller and target-training run identifiers are 401–405; neither failed controllers nor checkpoints are replaced after evaluation.

Evaluation and additional results. Ordinary evaluation uses the 42 ordered start–goal region pairs with two continuous initializations for each pair, giving 84 evaluation cases. We define sustained success as success on at least 80 of the 84 cases at three consecutive evaluations.

We perform five independent target runs. Before any target high-level learning, the transferred routing policy succeeds on all

$$84/84$$

evaluation cases in every run. It remains successful on all 84 cases throughout target fine-tuning.

The transfer method reaches the sustained-success criterion after 10,002–10,027 additional target interactions. Including the 25,200 interactions used to train the continuous navigation controller, this corresponds to

$$35,202\text{–}35,227$$

Run	Transfer additional	Transfer with lower	Standard SAC/HER	Saving	Final successes T / S / flat
401	10,027	35,227	60,000	24,773	168 / 144 / 166
402	10,016	35,216	55,000	19,784	168 / 144 / 168
403	10,006	35,206	45,000	9,794	168 / 104 / 166
404	10,002	35,202	60,000	24,798	168 / 104 / 168
405	10,011	35,211	60,000	24,789	168 / 144 / 168

Table 2: All five fixed-maze PointMaze runs. Interaction counts mark the third consecutive evaluation with at least 80/84 successes. The total for transfer includes 25,200 interactions for continuous-controller training; saving is Standard SAC/HER minus this total. The hierarchy without transfer does not attain sustained success within 50,000 target interactions. Final successes are out of 168; T and S denote the hierarchies with and without transfer.

training interactions.

Standard SAC/HER reaches the same sustained-success criterion after 45,000–60,000 training interactions. The hierarchy without transfer does not reach the criterion within its 50,000-interaction target-learning budget.

Across the five paired comparisons with standard SAC/HER, the reduction in training interactions is positive in every run, with median reduction

$$24,773.$$

A final evaluation uses four new continuous initializations for each of the 42 start–goal pairs, giving 168 cases. The transfer method succeeds on all

$$168/168$$

cases in every run.

These results show that the learned discrete policy can provide the high-level routing decisions immediately, while the continuous target uses its own controller to implement those decisions.

Evaluations are scheduled every 5,000 training interactions and do not update the learner. Evaluations before and after the change to target-value decisions share one interaction count and count once toward sustained success. Tab. 2 gives every run's attainment and final outcome. Fig. 7 shows the learning curves. The discrete source separately used 5,000 training moves and 112 evaluation moves; these are not continuous-control interactions.

C.3.2 PointMaze: transfer to unseen layouts

Problem definition. We next allow the maze structure itself to change. The target uses the same continuous PointMaze dynamics, actions, and reward as in Sec. C.3.1, but its free regions form a layout that was not used to train the source policy.

At the high level, each free region is a node in a graph, and the available semantic decisions are the legal neighboring directions. Unlike the fixed-maze experiment, the source policy cannot identify a route by memorizing a particular map. Instead, its decision uses the current and goal regions, their relative positions, the local graph structure, and how often neighboring edges have already been traversed.

The source policy is trained on four discrete layouts: a 12-region branched corridor, a 16-region ring, a 20-region layout with intersecting cycles, and an open 25-region layout. The source is trained for 16,384 discrete routing decisions. It receives reward 1 only when the goal is reached, with discount 0.99.

We first evaluate transfer on four new continuous layouts with 12, 16, 20, and 23 free regions. None of these four layouts is used for source training.

The map is a connected subset of a 7×7 grid with region centers $(c - 3, 3 - r)$. The source receives the full adjacency graph, current and goal regions, remaining decisions and per-direction

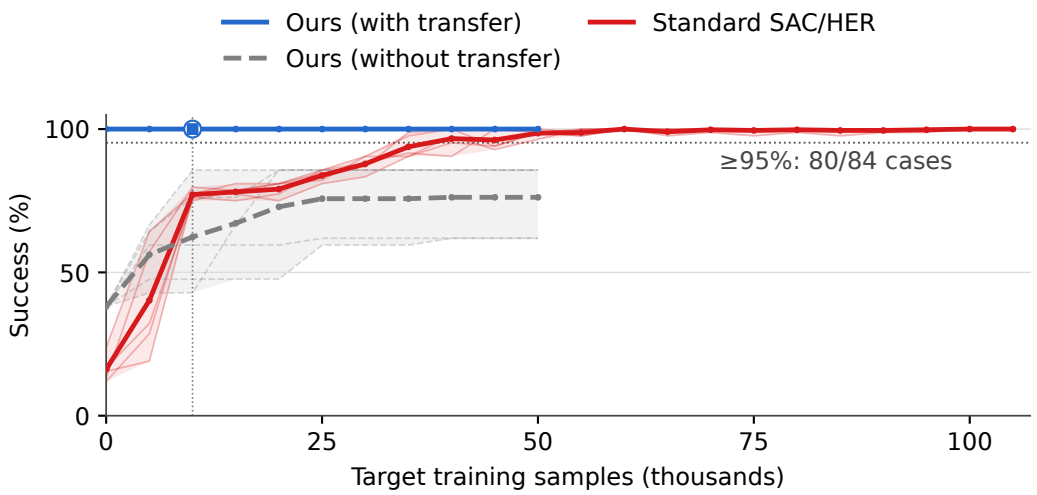


Figure 7: **Transfer reduces new target learning.** Five runs on 84 cases: thin traces, mean and range. The vertical line marks refinement after 10,000 interactions. Sustained success requires at least 80/84 at three consecutive evaluations. Both hierarchies use the same learned continuous controller. Hierarchy alone has no dominance guarantee over Standard SAC/HER because the learning procedures differ: without transfer it does not attain sustained success within 50,000 target interactions, while SAC/HER does so after 45,000–60,000. With transfer, all five runs attain it earlier than SAC/HER, even including the 25,200 controller-training interactions. The gain comes from the transferred routing policy. Source training and evaluation are counted separately.

traversal counts. Each node has eight features: current/goal indicators, two goal-relative coordinates divided by 4, and four traversal counts divided by 96. Source episodes have a 96-decision limit. Target execution has limits of 32 routing choices and 300 environment interactions. The target value function additionally receives two position offsets, two velocities, two final-goal offsets, and remaining interactions divided by 300. Reaching either episode limit closes the high-level transition; an external collection cutoff is not a completed skill.

Fig. C.4 illustrates source graphs and a distinct target graph. Fig. 5 shows the four target layouts and their learning curves.

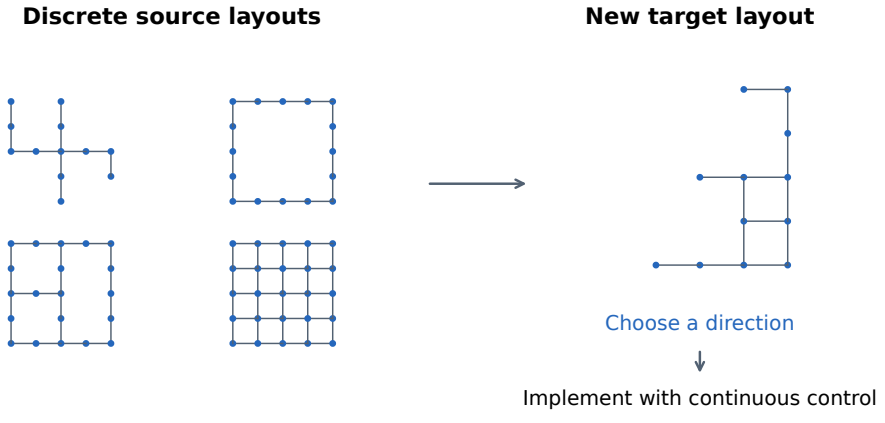


Figure C.4: A policy trained on several discrete graphs proposes a direction on a new graph. The continuous controller implements that direction. The graphs shown are taken from the recorded source and target layouts.

Method. We compare:

1. the hierarchy initialized with the learned source routing policy;
2. the same hierarchy without the learned source policy;
3. standard SAC/HER trained directly on each continuous layout.

The two hierarchical methods use the same target graph, the same legal neighboring directions, the same traversal history, and the same continuous navigation controller.

The source policy proposes a legal direction from the current graph state. To reduce repeated traversal, both hierarchical methods give preference to legal edges that have been traversed least often. Thus the two methods use the same target information; their difference is whether the initial routing proposal comes from the learned source policy or starts uniformly over legal directions.

The continuous navigation controller is trained once for 25,200 environment interactions on a five-region cross layout. It is then used on every target layout. Before each navigation call, the current region is translated to a common local coordinate system, so the same controller can implement a direction at different locations and on different layouts.

Unlike the tabular high-level learner in Sec. C.3.1, the target here learns a new neural high-level Q-function on each layout. This network estimates the value of each legal routing direction from the current target state. Both transfer and no-transfer methods start with fresh parameters, fresh replay data, and separate optimizers. No source value function is copied into the target.

Target learning alternates between collecting experience and comparing the current policy with a newly learned candidate. A candidate replaces the current policy only if, on the same comparison tasks, it does not reduce success and improves mean discounted return. The interactions used for these comparisons are included in the target-learning count.

Each hierarchy receives 30,000 target interactions for experience collection, in addition to the interactions used for policy comparison. Standard SAC/HER receives 50,000 training interactions.

Source episodes rotate among the four layouts and sample uniformly from their 132, 240, 380 and 600 distinct ordered region pairs. The source network uses sigmoid values, Huber loss, Adam at 3×10^{-4}, replay capacity 16,384 and batches of 64 sampled with replacement. Updates start at decision 260, occur every four decisions, and copy the target network every 128 updates, giving 4,032 updates. Random rotations and reflections are used during behavior and replay. Source collection prefers least-traversed directed edges and uses 0.1 uniform exploration over legal directions. Deployment combines eight rotations/reflections: its proposal is uniform over the directions receiving the most greedy votes. Both target methods restrict their proposal to legal edges with the least total traversal in either direction, renormalizing it or using a uniform distribution if no probability remains. Input counts remain directed, and only completed intermediate edges increment them. Both methods add 0.1 uniform-legal exploration during collection.

Translation changes the controller's position and both goal coordinates together, leaving velocity, world geometry, final goal and reward unchanged. Once the final-goal region is entered, the controller approaches the actual final goal without another routing-policy query. The shared controller uses the SAC/HER settings in Sec. C.3.1.

The target graph network has four message-passing rounds of width 32 and a 64-unit output head with sigmoid values. The paired learners begin with identical fresh parameters. Completed skills provide discounted rewards and continuation 0.99^{τ}, set to zero at either episode limit. Huber temporal-difference fitting maximizes over legal next directions. Replay capacity is 3,000, batch size 32, and Adam learning rate 3×10^{-4}. One update occurs per completed skill after 32 samples exist; the target network is copied after a completed collection episode with new updates. Incomplete prefixes count toward interactions but do not enter replay.

Policy comparisons occur at collection counts $250, 500, 1,000, 2,000, 3,000, 30,000$. Four paired task rollouts compare candidate and current policies; replacement requires complete observations, no aggregate success decrease and a mean discounted-return increase exceeding 10^{-12}. Ties retain the current policy. This is an empirical comparison, not a statistical improvement guarantee.

Standard SAC/HER uses the same four task conditions, with 57 inputs: position/velocity, achieved and desired goals, and the 49-region occupancy map. Its network, optimizer, replay and continuing-

task settings are those in Sec. C.3.1; it has no transferred policy, intermediate-goal controller or traversal-history rule.

Evaluation and additional results. Each new layout has four fixed start–goal tasks. For evaluation, each task is tested with five continuous initializations, giving 20 cases per layout.

For each of the four layouts, we perform three independent target-training runs, giving 12 paired comparisons between transfer and no transfer. We define sustained success as success on at least 19 of the 20 cases at three consecutive evaluations.

Before target learning, the transferred policy succeeds on all 20 cases in every one of the 12 runs. It remains successful on all 20 cases at every recorded evaluation.

The transfer method reaches the sustained-success criterion after 1,669–2,766 target interactions across the 12 runs. The hierarchy without transfer does not reach the same sustained criterion within its recorded learning window. The resulting conservative reduction in target interactions is at least

$$32,510$$

in every paired comparison.

Over the first 30,000 target interactions, the transfer method also has a larger area under the mean discounted-return curve in all 12 paired comparisons.

At the final evaluation, transfer remains successful on all

$$240/240$$

cases across the 12 runs. The hierarchy without transfer improves substantially during training, from 111/240 initially to 230/240 finally, showing that it can learn the new layouts, but requires substantially more target experience.

These four layouts establish that a routing policy learned across discrete source layouts can also guide continuous control on new maze structures. Sec. C.3.3 then evaluates the fixed procedure on 40 additional layouts.

The four tasks are both directions between the lexicographically first and last free regions and between the one-third and two-thirds indexed regions. The same task conditions are used in training and evaluation. Layout generation requires connectivity, a cycle, at least two regions with three or more neighbors, and diameter at least six, excluding previously used layouts under translation, rotation and reflection. Evaluation occurs at collection counts $0, 500, 1,000, 3,000, 5,000, 10,000, 15,000, 20,000, 25,000, 30,000$, with additional $40,000$ and $50,000$ evaluations for Standard SAC/HER. Initial and final reset/action streams coincide; other evaluations use separate streams. Each method has its own random-number generator. Reporting is detached from learning.

Sustained success is attained at the third distinct interaction count and must persist through subsequent evaluations. Return areas use recorded collection-plus-comparison interactions over 0–$30,000$, with linear segments and no extrapolation. Tab. 3 gives all twelve paired outcomes. Counting continuous-controller training as well, transfer has positive savings against Standard SAC/HER in nine pairs, but the 20-region median is $-2,717$. Standard SAC/HER has higher final return in all twelve pairs at its larger budget. These results do not establish uniform superiority over that method.

C.3.3 PointMaze: evaluation on 40 new layouts

Problem definition. After the four-layout experiment in Sec. C.3.2, we fix the source policy, continuous navigation controller, target-learning procedure, training budgets, and evaluation schedule. We then evaluate the same procedure on 40 additional PointMaze layouts.

The 40 layouts contain 12, 16, 20, or 23 free regions, with ten layouts at each size. Each layout is connected, contains at least one cycle, has at least two regions with three or more neighbors, and has graph diameter at least six. Layouts used previously for source training or method development are excluded, as are duplicates under translation, rotation, or reflection. No layout is selected using learning results.

Layout / run	Transfer steps	No transfer steps	Saving lower bound	SAC/HER steps	Saving with controller training	Return-area gain
0 / 0	2,207	$> 35,541$	33,334	50,000	22,593	0.197
0 / 1	1,716	$> 34,792$	33,076	25,000	-1,916	0.187
0 / 2	1,669	$> 34,179$	32,510	30,000	3,131	0.180
1 / 0	2,318	$> 36,666$	34,348	$> 50,000$	22,482	0.284
1 / 1	2,329	$> 36,312$	33,983	$> 50,000$	22,471	0.221
1 / 2	2,239	$> 36,352$	34,113	$> 50,000$	22,561	0.181
2 / 0	2,517	$> 36,793$	34,276	25,000	-2,717	0.246
2 / 1	2,546	$> 38,425$	35,879	25,000	-2,746	0.282
2 / 2	2,606	$> 37,683$	35,077	50,000	22,194	0.245
3 / 0	2,661	$> 38,770$	36,109	$> 50,000$	22,139	0.230
3 / 1	2,617	$> 37,777$	35,160	50,000	22,183	0.212
3 / 2	2,766	$> 37,991$	35,225	40,000	12,034	0.208

Table 3: All twelve unseen-layout comparisons. Costs are environment interactions for training plus policy selection. Transfer is already 20/20 successful initially; its listed cost marks the third confirming evaluation. Controller training adds 25,200 steps. Return-area gain compares the two hierarchies over the same paid 0–30k interval. Greater-than signs mark unobserved attainment; associated savings are conservative lower bounds. Layouts 0–3 have 12, 16, 20 and 23 regions.

Each layout uses four fixed start–goal tasks. Each task is evaluated from five continuous initializations, giving 20 evaluation cases per layout. Thus the 40 layouts, rather than the individual initializations, are the units used to compare performance across maze structures.

Method. The complete transfer procedure is unchanged from Sec. C.3.2. Each layout receives one transfer run and one matched run of the same hierarchy without transfer. Both methods use the same graph information, the same continuous navigation controller, and fresh target learning.

The transfer method begins with the learned source routing policy. The no-transfer method begins without that policy. Both then learn a fresh neural high-level Q-function that estimates the value of each legal routing direction from the current target state.

Each hierarchy receives 30,000 target interactions for experience collection, together with the interactions used to compare candidate policies. All of these interactions count toward target learning.

As in Sec. C.3.2, sustained success requires success on at least 19 of the 20 evaluation cases at three consecutive evaluations, followed by continued success at later evaluations.

For a layout where both methods reach this criterion, the target-step saving is the difference between their recorded attainment times. If the no-transfer method has not reached the criterion by its last recorded evaluation, that last interaction count gives a conservative lower bound on the saving. If the transfer method itself does not reach or maintain the criterion, the saving for that layout is left unresolved rather than assigned a numerical value.

We also compare learning performance over the first 30,000 target interactions using the area under the mean discounted-return curve. The area difference is divided by 30,000, so it can be interpreted as the average return advantage of transfer over this learning window.

Layouts are generated by uniformly sampling the sorted frontier of a growing connected set within a 5×5 interior, retaining the first admissible layout for each position in the evaluation set. The 40 layouts require 47 proposals. Reporting schedules and task selection are those in Sec. C.3.2. Each layout also receives one independent Standard SAC/HER run with 50,000 training interactions.

To bound the median without deleting unresolved layouts, each unresolved saving is assigned $-\infty$ only for this lower-bound calculation; it is not an observed performance value. The descriptive bootstrap resamples ten complete paired layouts within each size group, 10,000 times, and uses percentile endpoints. One paired training run per layout does not separate structural variation from training randomness.

Evaluation and additional results. Transfer has positive target-step savings on 35 of the 40 layouts. Three of these savings are observed directly, while 32 are conservative lower bounds because the hierarchy without transfer has not yet reached sustained success. The remaining five layouts have unresolved savings because the transfer run does not satisfy the complete attainment-and-retention condition.

Keeping all 40 layouts in the comparison, the median target-step saving is at least

$$34,246$$

interactions.

The return comparison is positive on all

$$40/40$$

layouts. The median normalized return-area improvement over the first 30,000 target interactions is

$$0.211.$$

The result is also consistent across maze sizes:

Free regions	Positive savings	Median saving lower bound	Median return-area gain
12	10/10	33,351	0.185
16	9/10	33,193.5	0.169
20	7/10	34,839.5	0.232
23	9/10	35,319	0.249

Table 4: Results by maze size, including all ten layouts in each group.

Thus every size group has a positive median saving and a positive median return-area improvement. A descriptive bootstrap over complete layouts gives a 95% interval of

$$[0.194,\ 0.242]$$

for the median return-area improvement.

Fig. 8 shows all layout-level comparisons. Tab. 4 retains every size group. Transfer initially succeeds in 786/800 cases and finishes at 794/800, versus 373/800 and 778/800 without transfer. Layout 13 does not attain sustained success; layouts 21, 22, 25 and 30 fail subsequent or final retention. All remain in the comparison.

Charging the full 25,200 controller-training interactions to each comparison with fresh Standard SAC/HER gives positive savings on 29/40 layouts, six negative differences and five unresolved cases. The 23-region median lower bound is -242. Standard SAC/HER has higher final discounted return in 38/40 comparisons. Thus the transfer result does not establish uniform end-to-end superiority. Discrete source training uses 16,384 routing decisions and is reported separately from continuous interactions.

For a practical illustration, Fig. 9 shows layouts 2 and 11 selected from the 40-layout set for positive transfer results and endpoint similarity under a rule allowing a return gap of at most 2.5% of the full plotted return range. These panels are selected illustrations, not a random sample. The main transfer claims use all 40 layouts.

C.4 Scenario III: transfer across divide-and-conquer domains

C.4.1 Merge Sort: learning divide-and-conquer structure

Problem definition. The source task is Merge Sort. A nonempty array of distinct entries is sorted by recursively solving two smaller subarrays and merging their sorted outputs.

Each non-base subproblem is divided at its midpoint. The policy chooses among four semantic skills:

$$\{\texttt{SolveChild1}, \texttt{SolveChild2}, \texttt{Combine}, \textsc{End}\}.$$

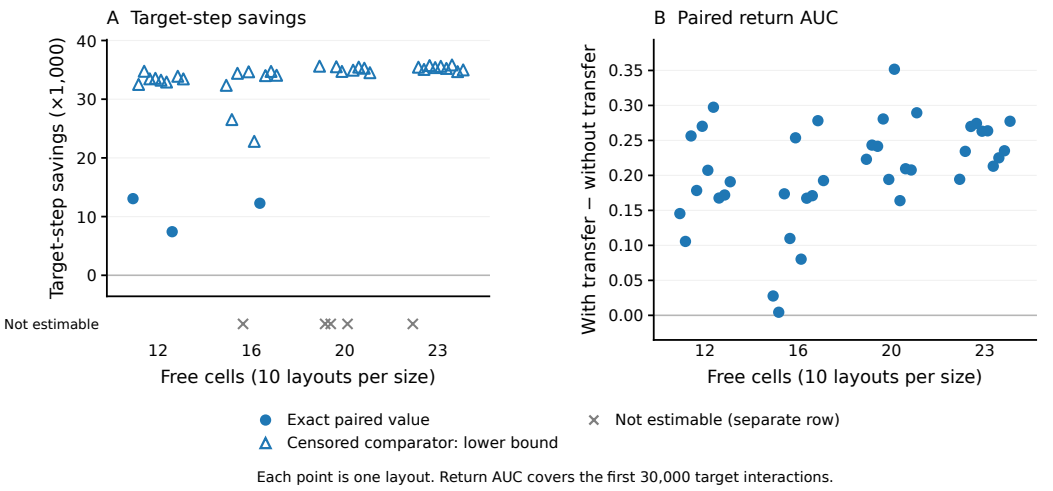


Figure 8: **Transfer across 40 additional maze structures.** Each point is one layout. Left: three savings are exact scheduled differences and 32 are lower bounds because the comparator has not attained sustained success. Five unresolved values appear in a separate nonnumeric row. Right: all 40 paired return-area differences over the first 30,000 target interactions. Horizontal offsets distinguish layouts without ordering by outcome.

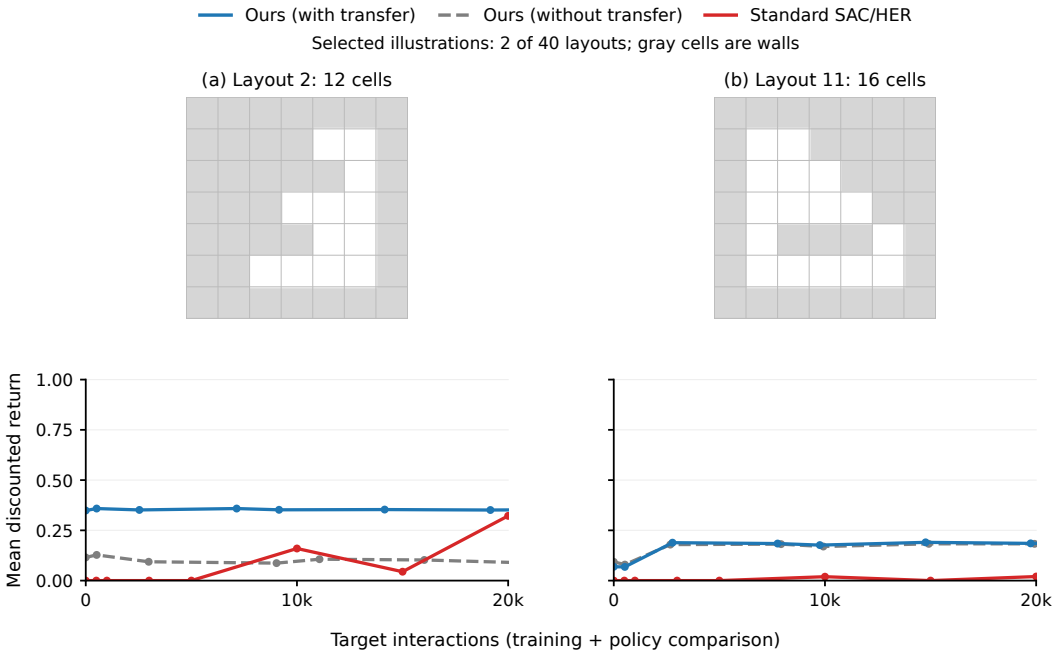


Figure 9: **Selected practical PointMaze comparisons.** Top: exact target layouts; gray regions are walls. Bottom: mean discounted return on 20 evaluation cases. Layouts 2 and 11 have positive transfer savings and return-area gains and were selected under the endpoint-display rule described in Sec. C.3.3. The main transfer claims use all 40 layouts; the comparison including controller training is reported in the same section.

Its semantic state is

$$z = (b_0, b_1, b_2),$$

where b_0 records whether the current problem is solved and b_1, b_2 record whether its two children are solved.

`SolveChild1` and `SolveChild2` recursively solve the corresponding child. `Combine` merges the two solved children, and END returns the completed result to the calling problem. Either child order is valid.

A valid completed root receives reward $+1$. Invalid execution or exhaustion of the command limit receives reward -1, and intermediate decisions receive reward 0. The discount is 1.

The environment stores the recursive stack, each frame's input array, child results, output and remaining command allowance. Any strictly increasing subarray is a base case, including arrays longer than one; a non-base array of length n splits after $\lfloor n/2 \rfloor$ entries. A solved frame is represented by 111. Selecting an already solved child, combining before both children are solved, ending before completion, or choosing a non-End command after completion ends the episode unsuccessfully. The command limit is 1,000. Each attempted command, including an invalid attempt or END, counts once; arithmetic operations are separate.

Fig. C.5 distinguishes choosing a decomposition from implementing it.

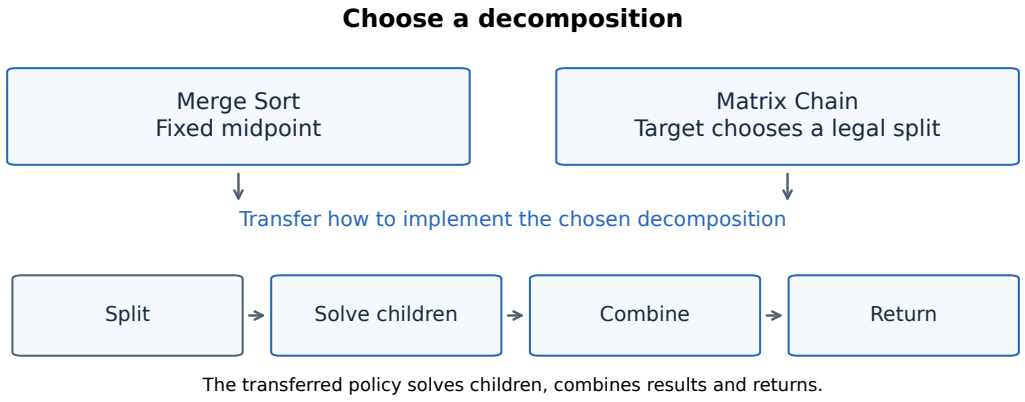


Figure C.5: Merge Sort fixes the midpoint; Matrix Chain has several legal splits. The transferred policy implements the chosen decomposition by solving its children, combining their results and returning.

Method. The source policy is learned from experience rather than constructed by hand. Training uses arrays of lengths 2, 4, and 8, with 3,000 training episodes in total. The policy uses step size 0.15 and exploration probability 0.2.

After training, the learned policy is queried at the five semantic states that occur during valid recursive execution. Its decisions are

$$
\begin{aligned}
000 &\mapsto \texttt{SolveChild2}, \\
001 &\mapsto \texttt{SolveChild1}, \\
010 &\mapsto \texttt{SolveChild2}, \\
011 &\mapsto \texttt{Combine}, \\
111 &\mapsto \textsc{End}.
\end{aligned}
$$

This policy contains the recursive relationship "solve the children, combine their results, and return." It does not contain a Matrix Chain split location or any information about matrix multiplication cost.

Training uses 200 episodes at length 2, then 800 at length 4 and 2,000 at length 8, with shuffled distinct entries. Values start at -2. After each root finishes, the final success/failure reward updates every stored semantic state/action using step size 0.15; intermediate child completion has no separate reward. The fixed final source run has identifier 2101. Only its categorical semantic policy is exported, not its numerical values.

Evaluation and additional results. The final source policy is evaluated on 256 shuffled arrays at each length from 2 through 8, for a total of

$$1,792$$

evaluation arrays. It successfully sorts all 1,792 arrays.

This learned recursive policy is the object transferred to Matrix Chain in Sec. C.4.2.

Evaluation uses a fixed shuffle stream (identifier 840131); repeated arrays at small lengths remain included. Source training uses 13,817 counted commands, with 13,449 more for evaluation, totaling 27,266; extracting the five decisions requires five policy queries. These source costs are separate from target learning. The same learned source is used by all five target comparisons.

C.4.2 Matrix Chain: transferring implementation of a given decomposition

Problem definition. Matrix Chain considers a product

$$A_i A_{i+1} \cdots A_j,$$

where matrix A_t has dimensions $p_{t-1} \times p_t$. In this experiment, the recursive decomposition is fixed before execution. The task is to solve the resulting subchains, combine their products, and return the completed root product.

For a selected split k, the two children are

$$A_i \cdots A_k \qquad \text{and} \qquad A_{k+1} \cdots A_j.$$

The available semantic skills are

$$\{\texttt{SolveChild1}, \texttt{SolveChild2}, \texttt{Combine}, \textsc{End}\}.$$

A valid `Combine` requires both child results. END returns a completed subproblem to its caller, or finishes the root problem.

A successful root return gives reward $+1$. An invalid decision or exhaustion of the command budget gives reward -1, and all other decisions give reward 0. The discount is 1. Multiplication cost is recorded separately and does not enter the execution-learning reward.

Execution is implemented by an exact shape-and-cost simulator, without multiplying dense numerical arrays. Each frame stores its dimensions, split, starting index, calling child position, two child results and product result; the full state includes the stack and remaining decisions. Single-matrix children are available immediately at zero cost. A valid child choice opens an unsolved multi-matrix child. Combining adds $p_{i-1}p_kp_j$ to the child costs and returns the product shape. Invalid choices are the same as in Sec. C.4.1.

An episode permits eight executed commands; near a scheduled training boundary it receives only the remaining smaller allowance. Success on the last allowed command takes precedence over exhaustion. Initiating or resuming a child at a higher level is not charged a second time; its executed commands are counted individually. Arithmetic operations and decomposition choices are separate.

Method. We compare:

1. the hierarchy initialized with the learned Merge Sort policy;
2. the same hierarchy without transfer;
3. standard Q-learning on the same recursive execution task.

All three methods receive the same Matrix Chain decomposition, the same information about which subproblems have been solved, and the same four semantic skills. The transfer method begins with the learned Merge Sort policy. The hierarchy without transfer must learn the same recursive decisions from Matrix Chain experience.

Target values start from scratch. No Merge Sort Q-values are copied into Matrix Chain.

Each executed choice among `SolveChild1`, `SolveChild2`, `Combine`, and END counts as one target decision. Each method receives at most 960 target-training decisions.

The evaluation set contains twelve fixed Matrix Chain problems: three two-matrix cases, four three-matrix cases, and five four-matrix cases. Evaluation occurs before target learning and then every 80 target decisions.

For each evaluation problem, we compute the exact success probability under the current learned policy. We use success probability at least 0.95 on every one of the twelve problems as the learning criterion.

The evaluation problems are all twelve decompositions of the six contiguous roots of dimensions $(3, 2, 4, 2, 3)$. Training splits are sampled uniformly according to subproblem size; the execution experiment does not train split optimization. Values use the structural stack (matrix count, split, solved-state information and calling child position) and remaining horizon, without dimension magnitudes or cost labels.

Both hierarchies can start a skill in the current problem or continue an unfinished child computation until its result returns to the corresponding ancestor. These choices do not select splits. The two hierarchies have the same value-learning and policy capacity. Without transfer, semantic preferences are learned from successful target root traces; failures still update values. The semantic policy is fixed within a root, and a selected child computation fixes its lower policy until return. Both levels' reward records extend to that return, with continuation at the calling problem and zero continuation only at root termination. Values used for continuation are saved before the corresponding updates. Root completion publishes the updated values and successful-trace counts. Evaluation makes no updates.

For the first 80 executed commands, each hierarchy uses its own semantic preferences with exploration probability $1/5$. Thereafter, exploitation and the learning target maximize over visited choices and preferred untried skills; all four skills are preferred when no semantic preference has been learned. Preferred maximizers win ties uniformly. Other ties use action order if preferences exist and uniform choice otherwise. Continuing an ancestor's child computation requires a previous observation before greedy selection; exploration includes every available choice. All methods use step size $1/N$ for visit count N. Standard Q-learning uses exploration $1/5$ from the start, ordinary online one-step updates, and no shared semantic preferences or higher-level value function.

Exact evaluation includes every branch of the current policy, including invalid choices. Mean-success area is the equally weighted average over twelve cases, integrated by linear segments over recorded counts $0, 80, \ldots, 960$ and divided by 960. Training split/action random-number identifiers are $510100000 + 100000i + 1000m + \{0, 1\}$ for run i and method m; detached sampled-evaluation identifiers are $520100000 + 100000i + 1000b + 10c + m$ for evaluation b and case c.

Evaluation and additional results. We perform five independent target-training runs. The transferred policy has success probability

$$1$$

on all twelve evaluation problems before any Matrix Chain target learning, and remains at probability one throughout all recorded evaluations.

The hierarchy without transfer first reaches at least 0.95 success probability on every problem after

$$80\text{–}320$$

target decisions across the five runs, with median

$$240.$$

Standard Q-learning does not reach this criterion within

$$960$$

target decisions in any of the five runs.

The transferred method also has a larger area under the mean success-probability curve than both comparison methods in every run.

Thus the Merge Sort policy transfers the divide-and-conquer execution relationship to Matrix Chain even though the two tasks use different domain operations. The target can execute a given Matrix

Replicate	Without transfer Threshold	Without transfer Area	Standard Q Area	Standard Q Final minimum
0	160	0.93788	0.50146	$1/3072$
1	320	0.88377	0.48514	$1/1536$
2	80	0.95900	0.49605	$1/6144$
3	240	0.87803	0.48685	$1/3072$
4	240	0.86535	0.48604	$1/6144$

Table 5: All five runs of the recursive-execution comparison. Transfer meets the criterion at zero and has normalized success area and final minimum probability one in all five; the hierarchy without transfer also finishes at one. Standard Q-learning never meets the criterion through 960 commands. Areas average all twelve cases at each scheduled evaluation and integrate the recorded command counts.

Chain decomposition immediately without first relearning when to solve children, combine their results, and return.

Fig. 6 shows the learning curves; Tab. 5 gives all five runs. Standard Q-learning finishes with success probability one on the seven two- and three-matrix cases, but not on the four-matrix cases. The standard-method comparison concerns the complete learning procedures; it does not isolate the effects of hierarchy and value estimation. Evaluation work is separate from the 960-decision training budgets.

C.4.3 Matrix Chain: separating implementation from decomposition choice

Problem definition. The previous experiment assumes that the Matrix Chain decomposition is given. Matrix Chain also contains a separate target-specific decision: where should each subchain be split?

For

$$A_i A_{i+1} \cdots A_j,$$

a legal split is

$$k \in \{i, \ldots, j-1\},$$

which creates the two children

$$[i, k] \qquad \text{and} \qquad [k+1, j].$$

If their products are combined, the final multiplication at this node costs

$$p_{i-1} p_k p_j$$

scalar multiplications.

Different legal splits therefore produce different recursive decompositions and can have different total multiplication costs.

Method. We separate this target-specific split decision from the transferred execution policy.

For each current subproblem, a higher-level target policy may choose among its legal split locations. Once a split has been chosen, the transferred divide-and-conquer policy from Sec. C.4.2 determines how that chosen decomposition is executed: solve its two children, combine their results, and return.

Each child subproblem chooses its own split independently. Thus choosing one split does not determine the entire decomposition tree.

Merge Sort has only one midpoint split, so its source task does not provide a preferred Matrix Chain split. The transferred policy therefore provides the recursive execution relationship, not a target-specific decomposition.

Exact dynamic programming gives the optimal Matrix Chain cost through

$$C^*(i, j) = \min_{i \le k < j} \left\{ C^*(i, k) + C^*(k+1, j) + p_{i-1} p_k p_j \right\}, \qquad C^*(i, i) = 0.$$

These optimal costs are used only for evaluation. They are not given to the learned split selector.

Evaluation and additional results. The transfer result reported in Sec. C.4.2 concerns execution of a given decomposition. The separate split-selection problem asks a different question: whether target learning can choose decompositions with lower multiplication cost.

The transferred Merge Sort policy does not provide an optimal split, an optimal cost-to-go, or an optimal-split label. Separate decomposition-learning experiments do not show an additional multiplication-cost improvement, because split selection is new target-specific knowledge for the three methods considered here. The resulting performance patterns are therefore similar, so we omit those additional results here.

This distinction is important: the demonstrated transfer is the reuse of the learned divide-and-conquer relationship, while decomposition choice remains a separate target-learning problem.

D SUPPORTING EXAMPLES

This appendix gives two supporting examples of the semantic-transfer framework. Sec. D.1 considers reuse across related tasks under partial observation. Sec. D.2 considers a discrete semantic policy implemented through continuous control.

D.1 BABYAI — SCENARIO I SUPPORTING EXAMPLE: SEMANTIC REUSE UNDER PARTIAL OBSERVATION

Problem definition. We use the BabyAI Unlock task (Chevalier-Boisvert et al., 2018) to test whether the key–door decisions learned in MazeBase+ remain useful when the target is partially observed.

The target agent receives a local visual observation, its current direction, and the mission description. The task requires finding a key, picking it up, reaching the corresponding door, and opening the door.

We reuse the same five semantic decisions as in the MazeBase+ source: navigate to the key, pick up the key, navigate to the door, open the door, and finish. The source policy therefore determines the high-level sequence, while BabyAI uses its own actions and observations to implement those decisions.

Method. The transferred high-level policy is the learned MazeBase+ policy from Scenario I. At each semantic state, it assigns probabilities to the five semantic decisions. No BabyAI target learning is performed.

Navigation in BabyAI uses a separate history-based controller. It acts only from the agent's observed images, direction, mission text, and its own action and observation history. It does not receive the hidden map or hidden coordinates.

Pickup and door opening use the corresponding BabyAI actions. The high-level policy receives task-progress indicators that identify the corresponding MazeBase+ semantic state.

We compare three high-level selectors:

1. the transferred MazeBase+ policy;
2. uniform random selection among the semantic decisions;
3. a constructed reference that uses the same task-progress information.

The uniform selector is included as a simple reference, not as a learned baseline.

Evaluation and additional results. We evaluate 64 BabyAI missions. For the transferred policy and the uniform selector, we use five independent selector runs per mission, giving 320 evaluations for each method. The constructed reference is evaluated once per mission.

The transferred policy succeeds in

$$315/320$$

evaluations. The uniform selector succeeds in

$$56/320,$$

while the constructed reference succeeds in

$$63/64.$$

Tab. 6 summarizes these results.

High-level selector	Successful evaluations
Transferred MazeBase+ policy	$315/320$
Uniform random selection	$56/320$
Constructed reference	$63/64$

Table 6: BabyAI success without target learning. The transferred and uniform selectors each have five runs per mission; the constructed reference has one. All methods use the same 64 missions.

The transferred policy and the constructed reference succeed on the same 63 of the 64 missions. Thus the learned MazeBase+ policy reaches essentially the same mission-level coverage as a hand-constructed semantic selector, without any BabyAI high-level training.

This experiment therefore supports portability of the learned semantic sequence rather than superiority over a constructed reference. It does not test learned language grounding: the correspondence between the MazeBase+ semantic decisions and the BabyAI task is provided through task-progress indicators. The result shows that the learned key–door sequencing policy can still be used when navigation must be implemented from partial observations and history.

D.2 Waypoint — Scenario II supporting example: ordered semantic decisions with continuous implementation

Problem definition. Each waypoint mission gives a start position and two ordered goals

$$g_1, g_2 \in [-1, 1]^2.$$

The agent must reach g_1, then reach g_2, and then finish the task.

The high-level policy chooses among three semantic decisions:

$$\{\texttt{Navigate}(g_1), \texttt{Navigate}(g_2), \textsc{End}\}.$$

Both navigation decisions remain available throughout the task. Progress advances only when the agent physically reaches the next required waypoint, within distance 0.08. A premature End fails.

This setup allows the same ordered decision policy to be used with different lower-level implementations: a discrete navigator in the source and continuous control in the target.

Method. For each of five independent runs, we first learn the three-decision high-level policy using a discrete navigation implementation. Its high-level state records only progress: whether neither waypoint, only g_1, or both waypoints have been reached.

The learned source policy is then combined with an independently trained continuous navigation controller. The semantic decisions remain $\texttt{Navigate}(g_1)$, $\texttt{Navigate}(g_2)$, and End; only the way navigation is implemented changes.

Target high-level values start from scratch. We compare the transferred hierarchy with the same hierarchy without transfer. Both receive the same goal coordinates, progress information, and continuous navigation controller.

For evaluation, source training missions, target training missions, and held-out missions are three distinct sets of 128 missions each.

Evaluation and additional results. Before any target high-level learning, the transferred policy succeeds on all

$$128/128$$

held-out missions in each of the five runs.

The hierarchy without transfer begins at

$$0/128$$

successful held-out missions. Its first recorded evaluation with $128/128$ success occurs after 251–501 target environment interactions across the five runs. All later evaluations are successful for both methods.

Fig. 10 shows this difference both in target environment interactions and in completed target episodes.

Thus the discrete source policy can immediately provide the correct ordered semantic decisions when those decisions are implemented by continuous control. The advantage is in initial target competence: after additional target learning, the hierarchy without transfer reaches the same success level.

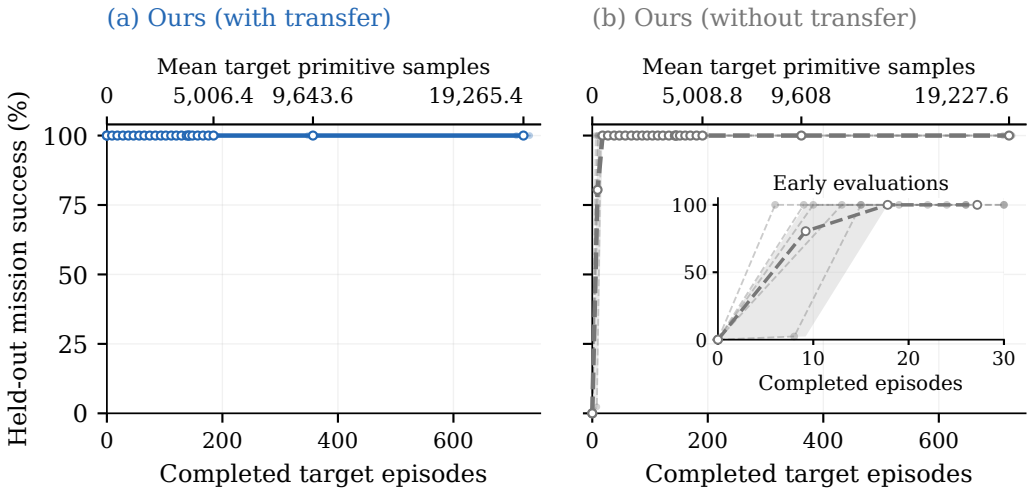


Figure 10: **Waypoint transfer provides immediate target competence.** Success on the same 128 held-out missions across five independent runs. Transfer succeeds before target learning; without transfer, the first recorded complete success occurs after 251–501 environment interactions. Both methods succeed at every later evaluation. Thin curves show individual runs, thick curves their means, and shading their range. Bottom axes show completed target episodes; top labels give mean environment interactions at the corresponding evaluation events for each method, including interactions in unfinished episodes. The inset shows early learning without transfer. Evaluation does not update training. Lines connect recorded evaluations.

E ADDITIONAL VISUALIZATIONS OF THE RETAINED INTERFACES

Fig. 11 shows continuous trajectories implementing discrete routing decisions.

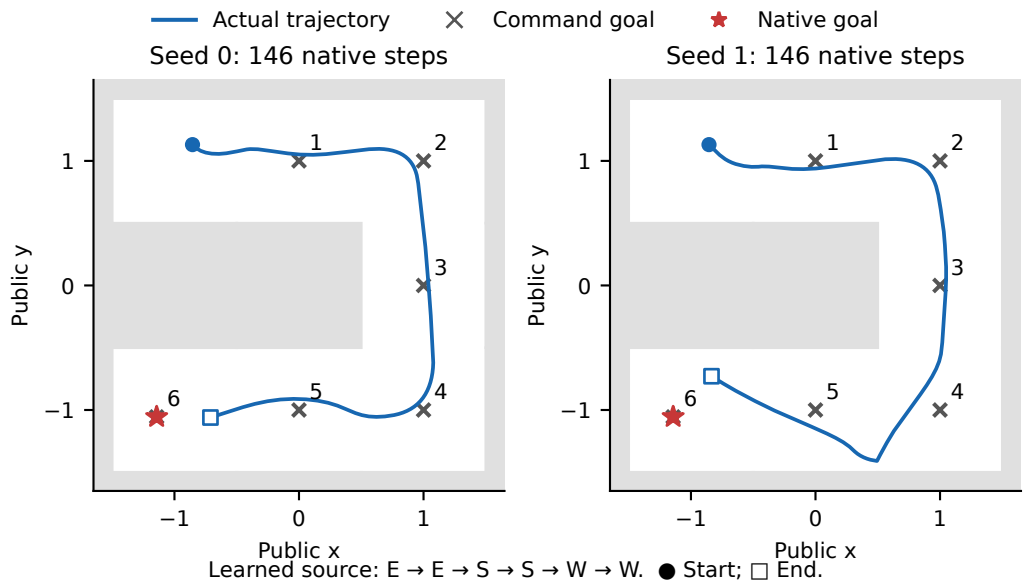


Figure 11: **Discrete semantic decisions realized by continuous trajectories.** Recorded executions from two acquired PointMaze controllers illustrate the source policy's direction sequence, intermediate goal commands and continuous native motion. Both reach the goal in 146 native steps. These previously acquired controllers illustrate the interface; they are separate from the five-run confirmation. The arrival radius is 0.45.

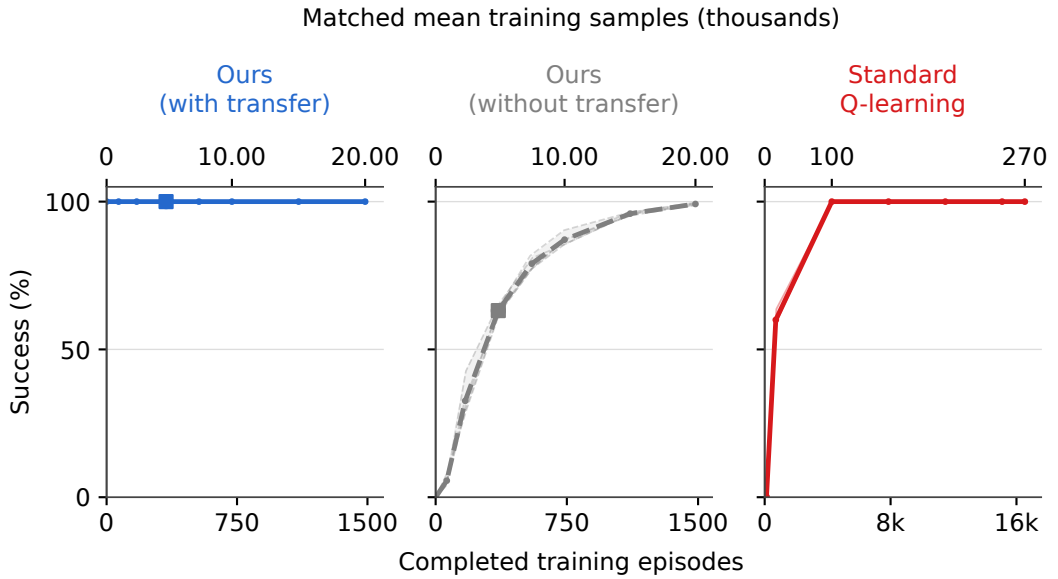


Figure 12: **Taxi on measured episode and sample clocks.** Each panel uses completed training episodes below and paired mean primitive counts above, from the same evaluation events. The mapping is method-specific. All five runs and complete budgets are retained. Circles/squares show before/after fine-tuning at 5k samples. Prior navigation, source and evaluation costs are excluded from both axes.

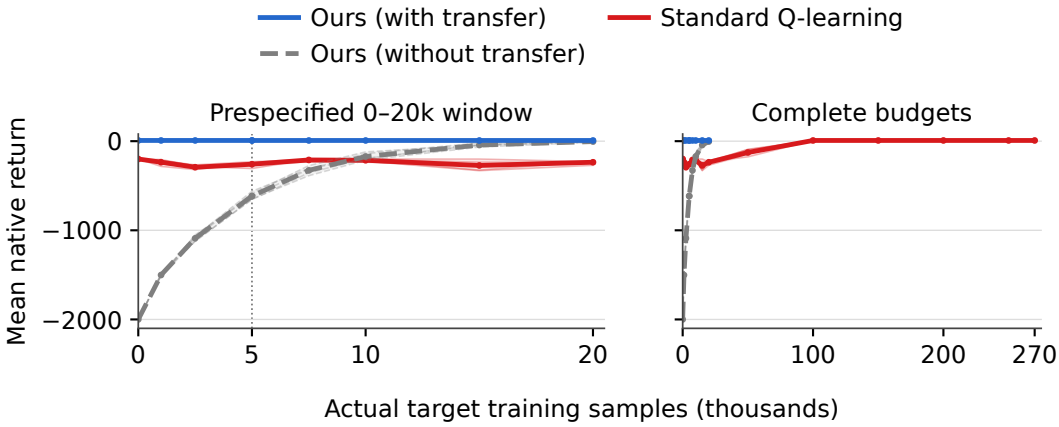


Figure 13: **Native Taxi return over all initial states.** The same evaluations as Fig. 12, with all five runs, mean and range. Native rewards exclude semantic training charges. The full return scale retains the low values during early learning; Fig. 12 uses success to make the three-method comparison readable.