Illustrated Landmark Graphs for Long-Horizon Policy Learning

Anonymous authors
Paper under double-blind review

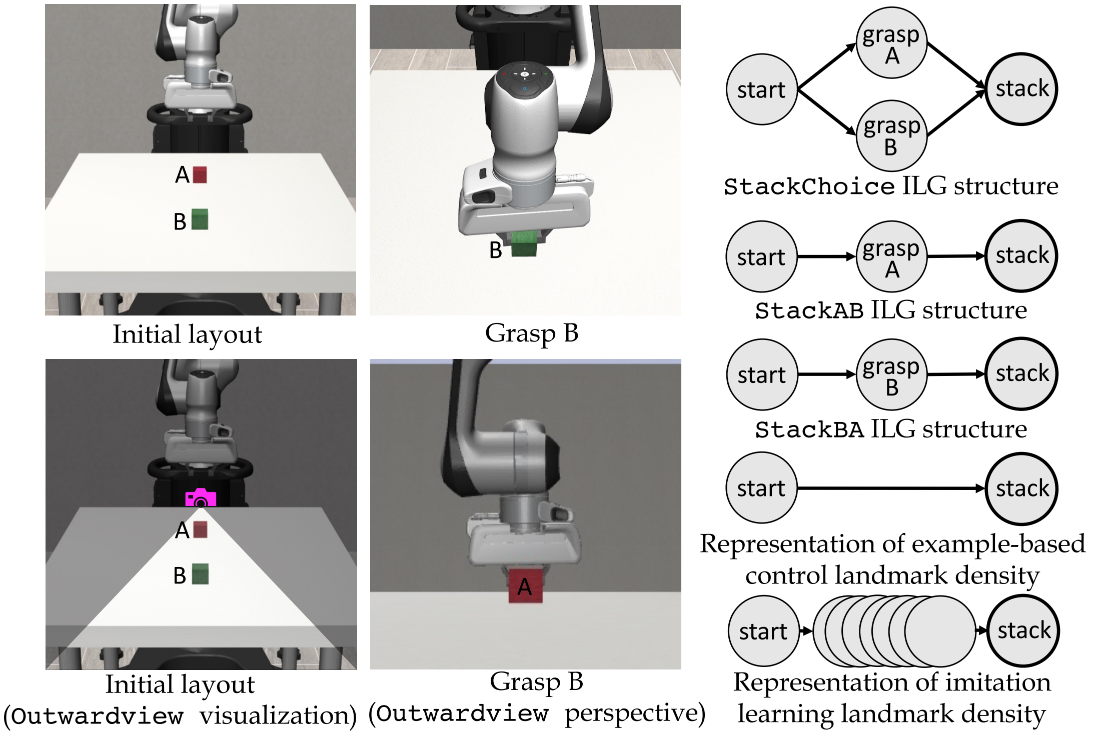
Abstract

To successfully apply learning-based approaches to long-horizon sequential decision making tasks, a human teacher must be able to specify the task in a way that provides appropriate guidance to the learner. The two most prominent policy learning paradigms, reinforcement learning (RL) and imitation learning, both require considerable human effort to specify a long-horizon task, either through dense reward engineering or providing many demonstrations that follow an approach that is feasible for the learner. We propose the *illustrated landmark graph (ILG)* as a form of task specification that exposes opportunities for the learner to customize its approach to its unique capabilities without the need for reward engineering, and allows the human teacher to intuitively provide intermediate guidance without the need for full-length demonstration. Each source-to-sink path in the ILG represents a way to complete the task, and each vertex along a path represents an intermediate *landmark*. To communicate the meaning of a landmark to the learner, the teacher provides *illustrative observations* drawn from states within the landmark. We further propose ILG-Learn, an approach that interleaves planning over the ILG, policy learning, and active querying of the human teacher to guide the learner. Our experimental evaluation shows that ILG-Learn learns policies that successfully complete a block stacking task and a 2D navigation task, while approaches that receive specifications in the form of final goal observations (RCE) or full-length demonstrations (behavior cloning) fail. Additionally, we show that a multi-path ILG allows ILG-Learn to adapt to the capabilities of a learner with limited perception.

1 Introduction

How can a human best teach an agent to perform a new long horizon task? The two most popular classes of approaches today are reinforcement learning (RL) and imitation learning. In RL, (Sutton & Barto, 2018) the teacher specifies the task via a reward function that assigns higher scores to more desirable environment configurations. The agent then interacts with the environment to learn how to achieve high rewards. For long-horizon tasks, the efficiency of this trial-and-error exploration depends intimately on the human teacher's ability to design a *well-shaped* reward function that encourages incremental progress towards the final goal (Laud, 2004; Ng et al., 1999; Sowerby et al., 2022; Gupta et al., 2022). As a concrete example, let us consider the block stacking task (`StackChoice`) shown in Figure 1. The task is to build a tower; the agent may choose to place block A (red) on block B (green) or *vice versa*. The task itself is fully specified by a simple "sparse" reward function that is 1 when the blocks are stacked and 0 otherwise. However, in practice, the human teacher must write a more sophisticated reward function that provides "dense" rewards during task execution. For example, it might incorporate small positive rewards for reaching various milestones towards the task: moving the gripper close to the block, grasping, lifting, aligning, and finally placing the block. To write such a reward function, the human teacher must be able to interpret the sensor readings available to the learner and prudently balance the weight given to each term in the reward function. In general, reward engineering is difficult, error-prone and requires considerable expertise, (Booth et al., 2023; Amodei et al., 2016; Skalse et al., 2022) limiting RL's applicability to long-horizon tasks.

Imitation learning (Hussein et al., 2017; Zare et al., 2023; Ravichandar et al., 2020) allows the human to teach the agent by demonstrating desired behavior. Compared to RL, imitation learning shifts the teaching



Initial layout

Grasp B

Initial layout (`Outwardview` visualization)

Grasp B (`Outwardview` perspective)

`StackChoice` ILG structure

`StackAB` ILG structure

`StackBA` ILG structure

Representation of example-based control landmark density

Representation of imitation learning landmark density

Figure 1: **Left:** Renderings of the `Stack` task, which we use as a running example throughout the text, including in our experiments. At the start of each episode, block A (red) starts closer to the robot than block B (green). Both blocks initialize at randomized positions along the centerline of the table. In the `Outwardview` condition, block A initially occludes block B. **Right:** ILG structure for variations of the `Stack` task, alongside representations of the landmark density used by baseline policy learning algorithms.

burden from reward engineering to demonstration. Although demonstrations are an intuitive form of task specification, providing high-quality demonstrations may not always be feasible (Ravichandar et al., 2020). The teacher might not know how best to perform a task, or how to manually operate the agent to do so, e.g. there may be no good interfaces to manually operate a biped robot to run smoothly. Even without these problems, demonstrations can be cumbersome to provide: in the `StackChoice` task, the teacher would need to teleoperate the robot and gather many demonstrations to adequately cover the states that the agent may encounter during deployment. Finally, demonstrations must account for the learner's limitations: e.g., stacking the blocks in one order may be easier for the robot to learn and perform than the other, because of limitations on its sensing, perception, or actuation abilities. These limitations are not always easily characterized in advance.

Reward functions and demonstrations occupy opposite ends of a design spectrum: a reward function specifies *what* must be achieved (perhaps without much intermediate guidance), while a demonstration specifies *how* to achieve it (perhaps too prescriptively). Ideally, the teacher could provide a specification that balances the declarative nature of a reward function (which exposes opportunities for the learner to customize its approach according to its capabilities) with the prescriptive nature of a demonstration (which provides fine-grained guidance). To address this gap (visualized in Figure 2), we propose the *illustrated landmark graph (ILG)* as a form of long-horizon task specification. Each vertex of an ILG represents an intermediate *landmark*, which is a subset of the environment's state space. To show the meaning of a landmark to the learner, the teacher must be able to provide a handful of *illustrative observations* drawn from states within the landmark. Each directed edge (u, v) in the ILG represents the *edge task* of transitioning the environment from the landmark represented by u to the landmark represented by v. Each sink vertex represents a *final landmark*. The long horizon task is to reach any final landmark, passing through intermediate landmarks on the way. Importantly, the ILG can contain multiple paths to a final landmark, exposing opportunities for the learner find a plan that suits its capabilities. Returning to our block stacking task, a human teacher can identify that grasping a block before placing it is a useful landmark and can communicate this to the learner via the `StackChoice` ILG shown in Figure 1.

To leverage the ILG for policy learning, we propose ILG-Learn. At a high level, ILG-Learn interleaves Dijkstra-style planning over the ILG with *example-based control*, (Fu et al., 2018; Singh et al., 2019; Eysenbach et al., 2021) a lightweight alternative to imitation learning that uses goal observations instead of full-length demonstrations, to learn an *edge policy* for each edge task. Beyond providing the ILG and its associated illustrated observations, the ILG-Learn teacher need only respond to binary success/failure queries during training to inform the learner's graph search. This intuitive teacher-learner interface allows the learner to benefit from exploration without requiring the teacher to engineer a reward function.

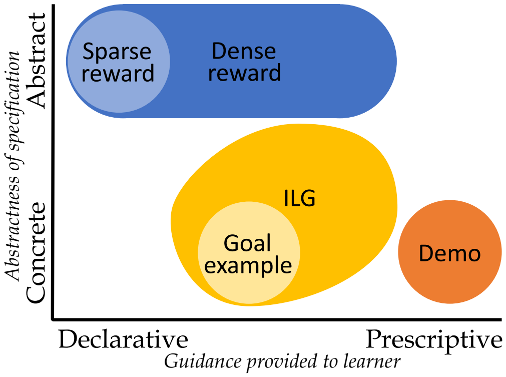


Figure 2: The ILG bridges the sparse, declarative guidance provided by goal examples and the dense, prescriptive guidance provided by demonstrations.

A key benefit of the ILG specification is the ability to represent multiple paths to overall success. The `StackChoice` ILG includes multiple paths to the final landmark ("*stack*")—depending on the learner's capabilities it may be easier stack the blocks in one order or the other. For example, if the agent were to perceive the world using an outward-facing camera mounted opposite the robot on the surface of the table (as shown in Figure 1), block B would initially be occluded, making it easier to start by grasping the (unoccluded) block A. By querying the teacher for success/failure feedback, the ILG-Learn learner can focus exploration along the most promising paths. Our experiments show that ILG-Learn adapts to the occlusion condition and learns a policy that grasps the unoccluded block first. This capacity for train-time adaptation reduces the burden on the human teacher to predict which approach will be most feasible for the learner.

Another benefit of the ILG is that the teacher can choose a *landmark density* (the number of intermediate vertices along a path in the ILG) that is dense enough to guide efficient exploration yet not so dense as to be overly prescriptive. At one extreme, the teacher could provide a landmark graph that comprises a single edge, thus recovering the free-form exploration of ILG-Learn's underlying example-based control algorithm. At the other extreme, the teacher could identify many intermediate landmarks to tightly scaffold exploration. The ideal landmark density depends on the task of interest; we believe that the intuitive structure of the ILG will enable the teacher to apply domain knowledge to select an appropriate landmark density without concerning themselves with the low-level details of the robot's perception and motor capabilities. Our experimental evaluation shows that ILG-Learn outperforms baselines that use end-to-end example-based control (which embodies minimal landmark density) and behavior cloning (which uses full-length demonstrations). In summary:

- We introduce the ILG as a form of long-horizon task specification that allows a human teacher represent multiple paths to success and provide intermediate landmarks without explicit knowledge of the learner's perception capabilities and without the need to provide full-length demonstrations of desired behavior.
- We propose ILG-Learn, a learning algorithm that leverages the ILG's interpretable-by-design nature to allow the teacher to provide lightweight guidance as the learner learns policies and a plan to suit its unique capabilities.
- We empirically show that providing a suitable ILG enables ILG-Learn to learn successful policies, while approaches that do not receive structured guidance (RCE and bahavior cloning) struggle. Additionally, we show that a multi-path ILG facilitates learning when the teacher cannot fully anticipate the learner's capabilities.

2 Related work

Hierarchical reinforcement learning and structured exploration. Hierarchical reinforcement learning (HRL) (Hutsebaut-Buysse et al., 2022; Pateria et al., 2021) decomposes a task into multiple subtasks that may be easier to learn.

Although an ILG specification does not fit precisely into the traditional *options* framework (Sutton et al., 1999) since the (1) the teacher does not explicitly define entry and exit conditions for each edge task and (2) the learner's choice of which edge task to execute is conditioned on the history of ILG edges traversed thus far, ILG-Learn is related to recent approaches (Araki et al., 2021; Neary et al., 2022; 2024) that start with human-specified options and simultaneously learn intra-option control policies and a policy over options. The main difference between ILG-Learn and these approaches is that ILG-Learn leverages example-based control and teacher-learner interaction to avoid the need for reward engineering and explicit specification in terms of environment states.

ILG-Learn's use of the ILG is directly inspired by DiRL's (Jothimurugan et al., 2021) use of the analogous *abstract graph*, which is built according to the structure of a temporal logic specification. The ILG-definable specifications are subsumed by LTL. The main difference between ILG-Learn and *specification-guided reinforcement learning* algorithms is that ILG-Learn uses human-in-the-loop interaction to guide the learner instead of an explicitly defined reward structure (see, e.g., Toro Icarte et al. (2022); Alur et al. (2022)).

Rather than reactively select a path through the ILG during execution, ILG-Learn uses the ILG as a scaffold for efficient exploration during training. During training, the learner freezes a single path through the ILG that is well-suited to the learner's capabilities; there is thus no high-level policy performing edge selection during deployment. This directed exploration towards the frontiers of the ILG is reminiscent of Go-Explore (Ecoffet et al., 2021). ILG-Learn differs from Go-Explore in that landmarks serve as the task specification (avoiding the need for a reward function), include human-defined relationships (in the form of the ILG), and are illustrated to the learner via teacher-provided observations (rather than reached organically).

Imitation learning and example-based control. As we argued in Section 1, the price of imitation learning's intuitive teacher-learner interface is rigidity: the learner is vulnerable to compounding deviations from the teacher's demonstrations (Ross et al., 2011) and may not enjoy the flexibility to adjust its approach to suit its own unique capabilities. To address these challenges, *inverse reinforcement learning* (Arora & Doshi, 2021; Adams et al., 2022) gleans the demonstrator's intent, allowing the agent to learn a policy through environmental interaction rather than rote memorization. In the context of long horizon policy learning, UVD (Zhang et al., 2024) and Relay Imitation Learning (Gupta et al., 2020) identify subgoals within demonstrations that can be used for compositional imitation inspired by goal-conditioned RL. Compared to ILG-Learn, these approaches remove the need for the teacher to identify landmarks, but in so doing reduce the teacher's ability to scaffold exploration and require the teacher to provide demonstrations rather than illustrative observations.

Example-based control (Fu et al., 2018; Singh et al., 2019; Eysenbach et al., 2021) allows the teacher to provide single-timestep examples of the environment state after task completion rather than full-length demonstrations. It is often significantly easier for a human to provide a single-timestep example than a full demonstration, and sparse guidance allows the learner more flexibility. However, for long-horizon tasks, such low landmark density may not provide the learner enough guidance to complete the task. At a high level, our proposed method, ILG-Learn, scales example-based control to long-horizon tasks by incorporating human exploration biases in the form of intermediate landmarks. We use the example-based control algorithm RCE (Eysenbach et al., 2021) to learn each edge policy that transitions the environment between landmarks.

Human-in-the-loop policy learning. Reinforcement learning with human feedback (RLHF) (Kaufmann et al., 2024; Arzate Cruz & Igarashi, 2020; Najar & Chetouani, 2021) elicits guidance from a human during training to reduce (or completely eliminate) reliance upon the environment's reward function. Some recent applications of RLHF to robotics query human preference over states to discover subgoals for HRL (Zhou et al., 2024) or goal-conditioned exploration (Villasevil et al., 2023). Instead of using human preference to discover landmarks, ILG-Learn requires the teacher to choose which landmarks to include in the ILG before learning begins. ILG-Learn then incorporates human feedback in the form of success/failure queries to direct exploration towards paths that cater to the learner's capabilities.

Human feedback can also be used to facilitate imitation learning. For example, HI-IRL (Pan & Shen, 2018) requires the human to provide subgoal states alongside a full-length demonstration. During training, the learner can ask for focused demonstrations of particularly hard-to-learn subgoals. Another recent approach, RLIF (Luo et al., 2024), builds upon the DAgger (Ross et al., 2011) family of algorithms by allowing a user to intervene during training to tell the learner when it behaves unsatisfactorily and provide a short demonstration of better behavior. The aforementioned techniques begin with a full-length demonstration of the task, which contrasts with our ILG's ability to express multiple paths to success. In a similar vein, Memarian et al. (2020) combine automata learning and inverse reinforcement learning to discover a subgoal decomposition alongside a policy to complete a long-horizon task. Their approach allows the learner to ask for specific guidance while discovering the subgoal decomposition but is not immediately applicable to continuous-state environments with difficult control tasks.

3 Illustrated landmark graphs

We introduce the *illustrated landmark graph (ILG)* as a form of long-horizon task specification. A key feature of the ILG is that the teacher communicates the meaning of each landmark to the learner using *illustrative observations* and does not need to provide explicit definitions in terms of the environment's state. In Section 3.1 we define the environment and policy model before defining the ILG specification format in Section 3.2. The ILG is a useful form of specification because of the way it facilitates teacher-learner interaction; we describe our proposed interaction interface in Section 3.3.

3.1 Preliminaries

As is standard in RL, we assume the agent interacts with an environment that can be expressed as a Markov Decision Process (MDP). An MDP comprises a continuous set of states $\mathcal{S}$, a continuous set of actions $\mathcal{A}$, dynamics governed by the transition probability distribution $p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$, and a starting state distribution η. Instead of specifying the task via a reward function (as is commonly done in MDPs), we will use an ILG as the task specification and incorporate human interaction to guide the learner.

At each timestep t, the agent receives an observation $o(\boldsymbol{s}_t) \in \Omega$ where Ω is a continuous observation space. The observation function $o : \mathcal{S} \to \Omega$ represents the learner's perception capabilities. In a real-world setting, $o(\boldsymbol{s}_t)$ may comprise raw sensor readings or the outputs of a pretrained perception module. A policy is a function $\pi : (\Omega \times \mathcal{A})^* \times \Omega \to \Delta(\mathcal{A})$ that maps a finite trace of observations and actions to a distribution over next actions.

3.2 ILG and satisfaction

Each vertex of an ILG represents a landmark, which is a subset of the state space $\mathcal{S}$, and each edge in the the ILG represents the *edge task* of transitioning between two landmarks. The ILG must have at least one sink vertex; each sink corresponds to a *final landmark.* The agent's goal is to reach a final landmark.

Formally, an ILG is a tuple (U, E, u_0, β) with vertex set U, directed edge set E, distinguished source vertex u_0, and landmark map $\beta : U \to 2^{\mathcal{S}}$ that maps each vertex to the set of states that comprise the corresponding landmark. An ILG must have at least one sink vertex. A trajectory $\xi = \boldsymbol{s}_0\boldsymbol{a}_0\boldsymbol{s}_1\boldsymbol{a}_1 \ldots$ satisfies the landmark graph (U, E, u_0, β) iff there exists a sink vertex $u \in U$ and a time t such that $\boldsymbol{s}_t \in \beta(u)$.

Figure 1 shows the structure of the `StackChoice` ILG. There are two source-to-sink paths: "*start* → *grasp A* → *stack*" and "*start* → *grasp B* → *stack*." Any trajectory that passes through a state in the landmark represented by the sink vertex "*stack*" satisfies the specification. The vertices "*grasp A*" and "*grasp B*" provide intermediate guidance to a policy learner. This guidance takes the form of teacher-learner interaction during training as detailed in the following subsection.

3.3 Interaction and illustration

As is common in RL, we assume that the learner does not begin with knowledge of the environment's transition probabilities; it must learn about environment dynamics through interaction. Similarly, we assume that the learner does not have explicit access to the ILG specification's landmark map $\beta : U \to 2^{\mathcal{S}}$. During training, the learner begins with access to the *structure* (U, E, u_0) of the ILG (U, E, u_0, β). In order to obtain guidance toward the landmark represented by a vertex u, the learner must interact with the teacher to request *illustrative observations* drawn from states within $\beta(u)$. The learner may also request success/failure feedback from the teacher to see if the current state of the environment lies within a particular landmark.

During training, the learner interacts with the teacher and with the environment using the following procedures. Training proceeds as a sequence of episodes. At each timestep t within an episode the learner receives an observation $o(\boldsymbol{s}_t)$ and may:

- `reset`() Reset the environment to a state $\boldsymbol{s}_0 \sim \eta$. The learner receives the observation $o(\boldsymbol{s}_0)$.
- `step`($\boldsymbol{a}_t$) Provide an action $\boldsymbol{a}_t$ to the environment, which causes the environment to transition to a state $\boldsymbol{s}_{t+1} \sim p(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$. The learner receives the observation $o(\boldsymbol{s}_{t+1})$.
- `requestIllustration`(u, ρ) where $u \in U$ is the vertex of the ILG to be illustrated and the second argument $\rho \in U^*$ lets the the learner tell the teacher which path it wishes to extend to u. In response, the teacher provides a dataset of illustrative observations $\mathcal{O}_u \subset \{o(\boldsymbol{s}) \in \Omega \,|\, \boldsymbol{s} \in \beta(u)\}$ of illustrative observations of states within $\beta(u)$. Although the path ρ does not affect the contract that the teacher must uphold, it may help the teacher choose a useful set of illustrative observations: in the `StackChoice` example, a good teacher would respond to `requestIllustration`(*stack*, *start* → *grasp A*) with examples of block A stacked on top of block B (not of B stacked on A).
- `querySuccess`(u) where $u \in U$ is a vertex of the ILG. The teacher provides binary success/failure feedback of whether the current environment state $\boldsymbol{s}_t$ is in the landmark $\beta(u)$.

Given an MDP $\mathcal{M}$, observation function o, and ILG $\mathcal{G}$, the learner tries to learn a policy π that maximizes the probability that a trajectory ξ drawn from π interacting with $\mathcal{M}$ while receiving observations according to o satisfies $\mathcal{G}$.

Importantly, the teacher is never asked to explicitly define the landmark map β. Instead, the teacher must be able to provide a set of illustrative observations for each landmark and serve as an oracle for `querySuccess`. The teacher can provide a set of illustrative observations by, for example, physically positioning a robot and allowing its sensors to perceive the environment. This intuitive interaction interface allows the teacher to guide the learner without needing to understand the low-level details of the learner's perception capabilities.

Underlying the ILG's usefulness is its customizability. To permit multiple approaches to complete the task, the human can include multiple paths that lead to a final landmark. In addition to using branching to expose options for high-level planning, the teacher can adjust the *landmark density* (the number of intermediate landmarks) along each path to provide the appropriate granularity of guidance. In the context of policy learning, low landmark density cedes resolution of low-level control details to the underlying learning algorithm. This is both convenient for the teacher and can result in well-optimized motion if the learner can successfully transition between landmarks. On the other hand, excessively low landmark density can render policy learning intractable, as the underlying learning algorithm is left with insufficient exploration bias.

4 Learning algorithm

We propose ILG-Learn, a human-in-the-loop approach to policy learning for ILG specifications. At a high level, ILG-Learn performs the following steps:

- **Policy learning.** For an edge (u, v) of the ILG, the learner uses example-based control to learn an edge policy $\pi_{(u,v)}$ that tries to transition the environment from the landmark represented by u to the landmark represented by v.

- **Success estimation.** Given a learned path policy (a sequence of edge policies), the learner queries the teacher for binary success/failure feedback to estimate the success probability of the policy.

- **Planning.** The learner tries to find a source-to-sink path of minimal cost, where cost is defined as the negative logarithm of the associated path policy's success probability.

At the outset, the learner receives the structure (U, E, u_0) of the ILG specification (U, E, u_0, β) but does not know how to reach any landmark. By interacting with the environment and the teacher (through the interface described in Section 3.3) learns policies to reach intermediate landmarks and a plan to satisfy the ILG specification by reaching a final landmark (represented by a sink vertex). ILG-Learn iteratively builds a tree of lowest-cost known paths from the source to the other vertices of the ILG. Since the cost of a path depends on the success probability of the learned edge policies, ILG-Learn interleaves graph search and policy learning to obtain the path costs on the fly during training. Learning proceeds as a series of *learning intervals*, each of which focuses on a single edge of the ILG. Once ILG-Learn finds the lowest cost path ρ to a sink vertex, ILG-Learn returns the plan ρ and the associated path policy.

Algorithm 1 sketches the full ILG-Learn algorithm and Figure 3 illustrates the steps of a learning interval. In the following textual description we will denote the ILG specification as (U, E, u_0, β) and assume the learner has access to the environment $\mathcal{M}$ and the human teacher through the interface described in Section 3.3. To make our description concrete, we will use the `StackChoice` task introduced in Section 1 as a running example. The structure (U, E, u_0) of the `StackChoice` ILG is shown in Figure 1.

Algorithm 1: ILG-Learn

```
Input:
    • ILG structure $(U, E, u_0)$
    • Access to MDP $\mathcal{M}$ via reset and step.
    • Access to human teacher via
      requestIllustration and querySuccess.
Output: Path $\rho$ and associated path policy
$\rho_{u_0} \leftarrow [\,]$;
$\pi_{u_0} \leftarrow$ None;
$reachProb_{u_0} \leftarrow 1$;
$reachProb_u \leftarrow 0 \quad \forall u \in V \setminus \{u_0\}$;
while selectEdge() returns $(u, v)$ do
    $\mathcal{O}(v) \leftarrow$ requestIllustration$(v, \rho_u)$;
    $\pi_{(u,v)} \leftarrow$ learnPolicy$(\mathcal{O}_v, \rho_u, \pi_u)$;
    $\pi \leftarrow$ sequencePolicies$(\pi_u, \pi_{(u,v)})$;
    $successProb \leftarrow$ estimateProbability$(v, \pi)$;
    if $successProb > reachProb_v$ then
        $reachProb_v \leftarrow successProb$;
        $\rho_v \leftarrow \rho_u \circ (u, v)$;
        $\pi_v \leftarrow \pi$;
$u \leftarrow \mathrm{argmax}_{u \in sinkVertices(\mathcal{G})}(reachProb_u)$;
return $\rho_u$, $\pi_u$
```

ILG-Learn parameters

illustrationCount: illustrative observations provided per request.

episodeLength: Fixed time horizon used when training each edge policy.

intervalLength: training episodes per learning interval.

estimationQueries: Number of rollouts (each with a success/failure query) performed at the end of each learning interval.

successThreshold: Lower bound on success probability for an edge to be considered learned.

intervalsLimit: Upper bound on learning intervals per edge.

Exploration order (`selectEdge`). ILG-Learn's exploration order is determined by the `selectEdge` subroutine. At the outset, ILG-Learn's tree of known paths contains only the "start" vertex u_0. To extend this tree, `selectEdge` chooses an edge leaving the tree for which to learn a control policy. ILG-Learn implements a fine-grained interleaving (mediated by the **successThreshold** and **intervalsLimit** parameters) of policy and plan learning to accelerate learning. For details, see Appendix A.

Let us suppose that `selectEdge` chooses the edge $(u, v) = (start, grasp\ A)$. This marks the start of a *learning interval* dedicated to learning the edge policy $\pi_{(u,v)}$.

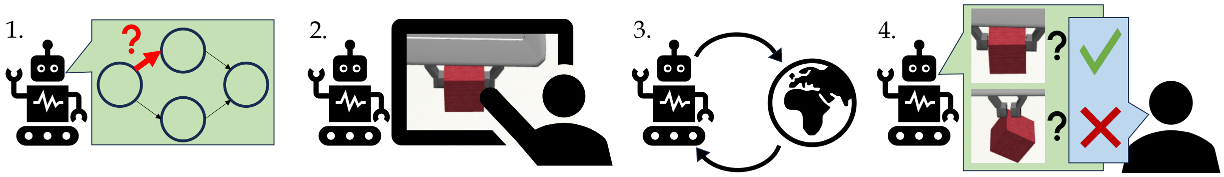


Figure 3: Stages of a learning interval: (1) The learner selects an edge and requests illustrative observations to guide learning. (2) The human teacher provides illustrative observations of the destination landmark. (3) The learner uses example-based control to learn an *edge policy* from environmental interaction. (4) The learner executes the edge policy and queries the teacher for success/failure feedback.

Human guidance (`requestGoals`). After selecting an edge (u, v), ILG-Learn asks the human teacher to provide a dataset $\mathcal{O}_v$ containing **illustrationCount** illustrative observations drawn from states within $\beta(v)$. In our example, the teacher would provide a set of observations that illustrate "*grasp A*".

Policy learning (`learnPolicy`). ILG-Learn runs example based control for **intervalLength** episodes each of **episodeLength** timesteps to learn an edge policy $\pi_{(u,v)}$. This learning process does not require interaction with the human teacher.

In our example, the learner would interact with the environment, trying to reach states that look similar to the human-provided examples of the "*grasp A*" landmark. Since this is the first edge along a path, each learning episode starts from the environment's initial state distribution.

In a future learning interval, ILG-Learn may try to learn the next step of the path, namely, how to stack block A on top of block B (assuming block A is already grasped). Since policy learning is sensitive to the starting state distribution, ILG-Learn needs to start learning $\pi_{(u,v)}$ from the distribution induced by the (already learned) path policy π_u. So in the case of learning to stack block A on block B, each episode would start by executing the $(\mathit{start}, \mathit{grasp}\ A)$ policy (for the same fixed horizon **episodeLength**) to reach the appropriate starting distribution for the $(\mathit{grasp}\ A, \mathit{stack})$ edge task.[1]

Success Estimation (`estimateProbability`) The last step of each learning interval is to estimate the probability that the learned edge policy $\pi_{(u,v)}$ successfully reaches $\beta(v)$. Just as it was important to start training $\pi_{(u,v)}$ from the state distribution reached by executing π_u, it is important to evaluate $\pi_{(u,v)}$ from the distribution reached by π_u. The `estimateProbability` subroutine executes **estimationEpisodes** additional rollouts of the sequenced policy, using `querySuccess`(u) at the last timestep of each rollout, to ask the human teacher whether the rollout was successful. If this empirical path probability exceeds that of the best known path ρ_v to v, ILG-Learn updates the best known path and associated path policy π_v. If the empirical probability exceeds **successThreshold** (or the per-edge max number of learning intervals **intervalsLimit** is reached) then ILG-Learn will consider the edge fully explored. At this point, the path cost (the negative logarithm of the success probability) is solidified. No more learning intervals can be allocated to (u, v). Future learning intervals can now be allocated to edges that leave v.

Termination. Training is complete when the `selectEdge` subroutine returns *None*, indicating that ILG-Learn has found the lowest-cost path in the ILG and the associated path policy. Otherwise, a new learning interval will commence, focusing exploration along the edge chosen by `selectEdge`.

5 Experiments

Our experiments show that ILG-Learn can learn policies to successfully complete long horizon tasks. To better understand the importance of allowing the teacher to choose the ILG's landmark density we evaluate ILG-Learn against RCE, a state of the art example-based control algorithm that does not use intermediate landmarks, and behavior cloning (BC), an imitation learning algorithm that receives full-length demonstra-

[1] In our experimental evaluation, we include the environment steps used during resets in the total environment steps used.

tions. Since the ILG task specification is new, there do not exist direct analogs of ILG-Learn to baseline against; rather, RCE's examples and BC's demonstrations represent two alternatives to ILG specifications.

We also show that ILG-Learn can discover the path through multi-path (branching) ILG that suits the learner's capabilities, reducing the burden on the teacher to predict the best approach to task completion *a priori*. We conduct our experiments in simulation using the following environments:

Stack. Our `Stack` family of environments is a customized `robosuite` (Zhu et al., 2020) environment that simulates a 7-DoF Franka Panda arm that receives 55-dimensional state observations and uses a 7-dimensional action space (which represents an operational space controller with fixed impedance). As introduced in Section 1, the learner must stack the blocks to build a tower. We include multiple versions of the task: `StackChoice` (stack the blocks in either order), `StackAB` (stack the red block A on the green block B), and `StackBA` (stack block B on block A). To study ILG-Learn's ability to adapt to a learner's unique capabilities, we further include the `StackChoice-Outwardview` and `StackBA-Outwardview` variants, which simulate observations collected from an object-detector operating from a camera mounted on the same side of the table as the robot arm. Whenever a block is occluded, we mask the corresponding components of the observation space. Figure 1 illustrates the `Stack` tasks; for more details see Appendix D.

Point Maze. We use custom layouts of the Point Maze environment from Gymnasium Robotics (de Lazcano et al., 2023; Fu et al., 2020). The agent is a force-actuated point-mass with a 2-dimensional action space and receives 4-dimensional observations that comprise its position and velocity. The agent must navigate from its starting position in the lower-left to the goal position in the upper-right. We include three variants of a diagonal maze (`DiagonalMaze3x3`, `DiagonalMaze5x5`, `DiagonalMaze7x7`) that differ in the length of the task. We also include a `DiagonalMaze7x7-Coarse` variant that has low landmark density and two variants of a 4x4 maze that differ only in their associated ILG specifications. Illustrations are shown in Figure 4.

5.1 Customizable landmark density

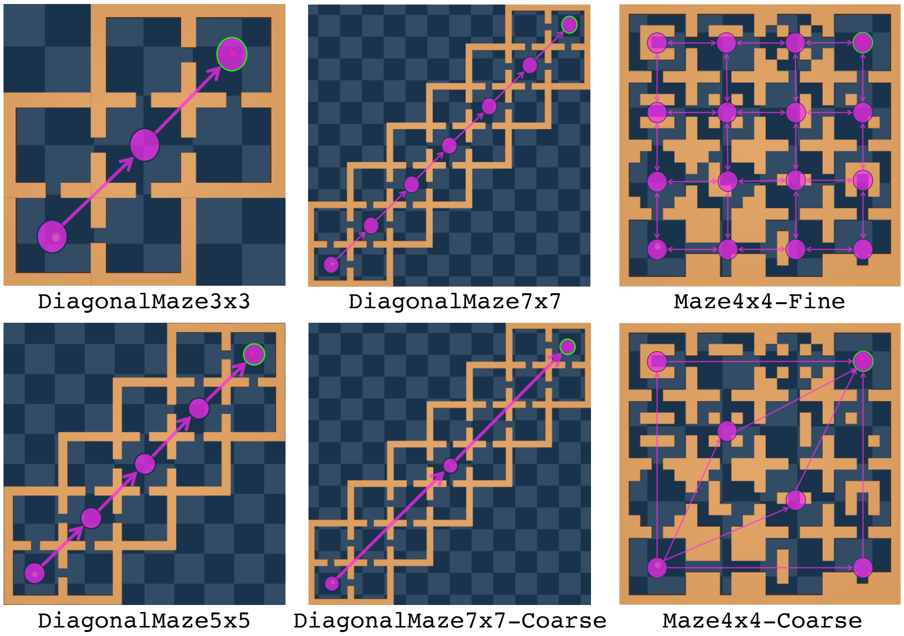
To disentangle the importance of allowing the teacher to choose the ILG's landmark density from the importance of multi-path specifications, we first turn our attention to tasks specified with a linear ILG. In the `StackAB` environment (Figure 1), the ILG-Learn learner receives a linear ILG that mandates placing the red block A on the green block B. The blocks are initially placed along the centerline of the table, with block A closer to the robot than block B. The exact positions of each block are randomized. For each landmark, the teacher provides 50 illustrative observations, for a total of 100 illustrative observations. At the end of each learning interval (100k environment steps, exclusive of steps used during resets) the teacher responds to 30 success/failure queries. To admit a fair comparison, the RCE baseline receives 100 goal examples, all of block A having been stacked on block B. We provide the BC baseline with 40 full-length demonstrations gathered using a scripted policy that includes some random variation (see appendix D for details). Each demonstration is 200 timesteps in length and completes the task in ∼150 timesteps. We allow ILG-Learn and RCE to train for 10,000 environment steps and allow BC to train until the training loss stops decreasing. Appendix E contains learning curves.

ILG-Learn achieves an average success rate of 0.752 for the `StackAB` task (Table 1), which is markedly higher than the success rates achieved by the RCE baseline (0.050) or the BC baseline (0.106). The fact that the RCE baseline achieves near-zero success rate shows that single-frame observations are too sparse to guide the learner towards success; allowing the teacher to specify an ILG with the intermediate "grasp A" landmark was crucial for ILG-Learn's success. On the other hand, the BC baseline, which receives very dense guidance towards the goal, yields policies that rarely succeed, showing that carefully mimicking a teacher's detailed demonstrations does not necessarily yield a successful policy.

The `DiagonalMaze` family of environments shows that the teacher's choice of landmark density is more important for tasks with longer horizons. `DiagonalMaze-3x3`, `DiagonalMaze-5x5`, and `DiagonalMaze-7x7` differ only in the length of the path that the learner must travel. In each `DiagonalMaze` variant, we provide ILG-Learn with 10 illustrative observation per landmark, the RCE baseline with 100 examples all drawn from the final goal region, and the BC baseline with 10 full-length demonstrations gathered by a scripted policy. Looking again at Table 1, we see that ILG-Learn consistently matches or exceeds the performance

	ILG-Learn (ours)	RCE	BC
StackAB	**0.752**	0.050	0.106
StackChoice-Outwardview	**0.952**	-	-
StackBA-Outwardview	**0.147**	-	-
DiagonalMaze3x3	**1.00**	0.00	0.990
DiagonalMaze5x5	**0.991**	0.00	0.826
DiagonalMaze7x7	**0.971**	0.00	0.476
DiagonalMaze7x7-Coarse	0.00	0.00	**0.476**
Maze4x4-Fine	1.00	**0.400**	-
Maze4x4-Coarse	**0.800**	0.400	-

Table 1: Final success rates (means over 5 trials). Both ILG-Learn and RCE are trained for a maximum of 10 million environment steps, see Appendix D for experimental details and Appendix E for learning curves. Entries marked with "-" indicate experiments that were not needed for our investigation.


DiagonalMaze3x3 DiagonalMaze7x7 Maze4x4-Fine

DiagonalMaze5x5 DiagonalMaze7x7-Coarse Maze4x4-Coarse

Figure 4: The Maze family of environments with ILGs superimposed (light pink). In each, the agent starts in the lower-left room and must navigate to the upper-right room. DiagonalMaze5x5 is defined analogously to the 3x3 and 5x5 variants. The Maze family of environments with ILGs superimposed (light pink). In each, the agent starts in the lower-left room and must navigate to the upper-right room. DiagonalMaze5x5 is defined analogously to the 3x3 and 5x5 variants.

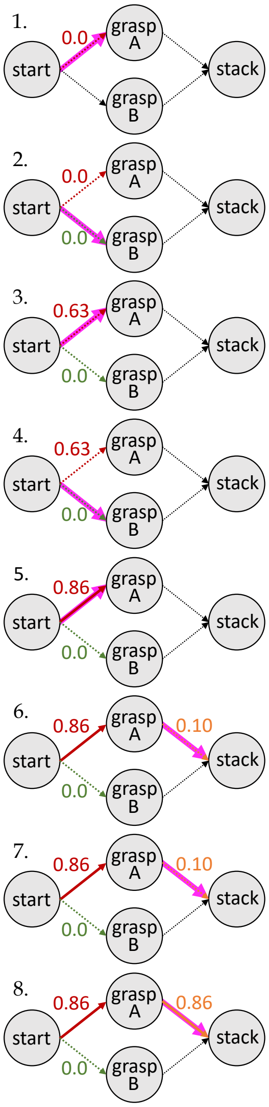


Figure 5: Learning intervals for StackChoice-Outwardview. The most recently explored edge is highlighted in pink and results of estimateSuccess are annotated.

of the RCE and BC baselines. Importantly, BC achieves high success rate for the relatively short-horizon DiagonalMaze3x3 but its performance deteriorates dramatically as the size of the maze increases. On the other hand, ILG-Learn consistently yields near-perfect success rates regardless of maze size. This supports the intuition that ILG-Learn can scale to long horizon tasks by (1) allowing the learner flexibility to explore and discover suitable policies to reach each landmark and (2) incorporating success/failure feedback during training to ensure each edge policy is learned successfully before moving on to train successive policies. This unlike BC and other imitation learning algorithms that are vulnerable to compounding deviations between the demonstrations and the actual trajectories taken by the agent.

To realize ILG-Learn's benefits, the human teacher must provide an ILG with appropriate landmark density. Given the excessively low landmark density of `DiagonalMaze7x7-Coarse`, ILG-Learn consistently fails to learn a successful policy (while the ILG of `DiagonalMaze7x7` yields near-perfect policies). Such a failure occurs when the exploration needed to learn an edge policy exceeds the capabilities of ILG-Learn's underlying example-based control algorithm. We provide advice for selecting landmark density in Section 6.

5.2 Multi-path specifications

A critical attribute of the proposed ILG specification is that it can include multiple source-to-sink paths, each representing an approach to complete the long-horizon task. To evaluate whether ILG-Learn can discover the path that best suits the learner's capabilities, we turn to the `StackChoice-Outwardview` task introduced in Section 1 and illustrated in Figure 1. The task is to stack the blocks (in either order). Importantly, the learner observes the position of each block via a simulated object detector based on a camera situated on the same side of the table as the robot, facing outward toward the blocks. Since the red block A starts closer to the camera than the green block B, the learner cannot detect the position of block B until a block is moved to break the occlusion.

As expected, in all five trials (random seeds) of ILG-Learn, the learner acquired a policy that follows the path "$\textit{start} \rightarrow \textit{grasp A} \rightarrow \textit{stack}$" achieving an average success rate of 0.952. Figure 5 illustrates the exploration process followed by the learner during one representative trial. Initially, the learner explores the ILG by alternating between allocating learning intervals to the $(\textit{start}, \textit{grasp A})$ edge as well as the $(\textit{start}, \textit{grasp B})$ edge. However, after 5 training intervals (about 700 thousand total environment steps) the learner successfully acquires a policy for the $(\textit{start}, \textit{grasp A})$ edge, and is thus able to reach the "*grasp A*" landmark. ILG-Learn's best-first search heuristic (detailed in Appendix A) guides further exploration along this path, focusing on the $(\textit{grasp A}, \textit{stack})$ edge. At the end of the 8^{th} learning interval (slightly more than 1 million total environment steps), the learner has acquired a policy that it estimates has a success probability of 0.86. Since this exceeds the user-specified **successThreshold** of 0.8, ILG-Learn terminates. Note that in reality, the success rate of the final policy is 0.947 (estimated from 1000 rollouts)- discrepancy between the learner's estimate of its own success probability can arise because (1) the learner only uses a modest number (in this case 30, the value of the **estimationQueries** parameter) of success/failure queries and (2) the learner only queries the final timestep of each rollout while we allow success to occur at any point along the trajectory.

The adaptive exploration scaffolded by the ILG described above is crucial for the ILG-Learn learner to successfully complete the task. To illustrate this, we compare against the `StackBA-Outwardview` task, which differs from `StackChoice-Outwardview` only in the ILG provided to the learner. `StackBA-Outwardview` forces the learner to start by grasping block B, which is very hard for the learner, since block B is initially occluded. As expected, ILG-Learn achieves a low average success rate of 0.147 on `StackAB-Inwardview`.

If the human teacher were able to predict which path would be easiest for the learner to follow, the teacher could simply provide a linear ILG (in the the above example, "$\textit{start} \rightarrow \textit{grasp A} \rightarrow \textit{stack}$"). However, such upfront prediction may pose a substantial burden to the teacher. A key benefit of using illustrative observations to specify landmarks is that the teacher does not need to be intimately familiar with the learner's perception capabilities—the same specification might even be used for multiple robots with different camera viewpoints. Moreover, using example-based control as a subroutine to let the learner acquire policies to achieve each landmark frees the teacher from having to worry about low-level details of the learner's motor capabilities. To fully realize these freedoms afforded to the teacher, the high-level task specification must also be, to some degree, agnostic to the intricacies of the learner's perception, control, and policy learning capabilities. A branching ILG provides this freedom to the teacher: the teacher can provide an ILG to scaffold exploration and allow the learner to interact with the environment to discover a plan that best fits its unique capabilities.

6 Discussion

In Section 5 we observed that ILG-Learn can learn successful policies for manipulation and navigation tasks. In this section, we discuss how the ILG definition affects the feasibility and efficiency of learning, providing both advice for users of ILG-Learn and highlighting directions for future work.

Landmark graphs and feasibility. To apply ILG-Learn, the human teacher must provide an ILG decomposition that contains at least one path that can feasibly be learned via iterated example based control. Although allowing the teacher to provide a branching ILG reduces the teacher's burden to predict the best landmark decomposition, some tasks may be hard for humans to effectively decompose. Future work could dynamically revise the ILG during training by prompting the teacher to provide new intermediate landmarks when the learner struggles.

Another future direction is to reduce the burden on the human teacher by prompting a foundation model to construct the ILG from natural language and image descriptions of the task and avoid the need for human-provided landmark examples, taking inspiration from works such as SayCan (Ahn et al., 2022) and VIP (Ma et al., 2023) while incorporating lightweight human feedback to ensure alignment with the teacher's intent.

Subtask difficulty and graph exploration. As established in Section 5, decomposing a long-horizon task into intermediate landmarks and providing a branching ILG that permits multiple paths to success allows ILG-Learn to solve long-horizon tasks. Applying ILG-Learn to a new task thus raises the questions "What is the appropriate landmark density?" and "How much branching should the ILG include?" At a minimum, the teacher should design an ILG that is coarse enough that each landmark is meaningful to the human eye (since the teacher must respond to success/failure queries). Beyond this, we recommend that the teacher err on the side of providing a dense ILG with many landmarks and a large amount of branching. The best-first search pattern introduced in Section 4 and detailed in Appendix A is designed to avoid exploring every edge in the ILG.

To gain empirical insight, we compared the performance and efficiency of ILG-Learn on a 2D navigation task using a "fine" ILG (`Maze4x4-Fine`) versus a "coarse" ILG (`Maze4x4-Coarse`). The tasks are visualized in Figure 4 and learning curves can be found in Appendix E. In this case, either ILG suffices to learn a successful policy: we turn our attention to how much efficiency we would lose by choosing the excessively fine decomposition of `Maze4x4-Fine`. Looking at the learning curves, it takes significantly more (comparing medians, 7x as many) total environment steps to learn a successful policy in `Maze4x4-Fine` than in `Maze4x4-Coarse`. Although this loss of efficiency is regrettable, it is preferable to accidentally providing too coarse an ILG and having learning fail altogether.

Moreover, we observed in our experiments with `Maze` tasks that many learning episodes for each landmark are spent learning how to smoothly decelerate the point-mass agent to avoid overshooting the landmark. Since we perform *tabula rasa* example-based control for at each edge, this learning effort is duplicated many times, proportional to the number of edges explored in the ILG. If future work develops an example based control algorithm that can adapt more quickly to novel tasks (perhaps by fine-tuning a pre-trained policy), we could replace RCE with that algorithm in ILG-Learn's `learnPolicy` subroutine to reduce the efficiency reduction introduced by an excessively fine ILG.

Human annotation burden. Finally, our `estimateProbability` subroutine relies on human annotation. For our experiments, we chose parameters such that the human responds to 30 success/failure queries per 100k training steps; for our `StackChoice-Outwardview` experiment this amounted to an average of 360 active queries per trial. While developing ILG-Learn, we tried to reduce this dependence by using a small amount of human-annotated data to train a binary success classifier for each landmark in a manner inspired by Singh et al. (2019). We found that even with hundreds of labeled positive examples gathered from successful executions of the policy being trained, there was often no threshold on the output of the classifier neural net that reliably separated success from failure. In the future, we plan to incorporate a pretrained vision-language model, perhaps augmented with residual network trained on in-domain data, to reduce the human annotation burden.

References

Stephen Adams, Tyler Cody, and Peter A. Beling. A survey of inverse reinforcement learning. *Artif. Intell. Rev.*, 55(6):4307–4346, aug 2022. ISSN 0269-2821. doi: 10.1007/s10462-021-10108-x. URL https://doi.org/10.1007/s10462-021-10108-x.

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 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 and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. *A Framework for Transforming Specifications in Reinforcement Learning*, pp. 604–624. Springer Nature Switzerland, Cham, 2022. ISBN 978-3-031-22337-2. doi: 10.1007/978-3-031-22337-2_29. URL https://doi.org/10.1007/978-3-031-22337-2_29.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016.

Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan Decastro, Micah Fry, and Daniela Rus. The logical options framework. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 307–317. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/araki21a.html.

Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103500. URL https://www.sciencedirect.com/science/article/pii/S0004370221000515.

Christian Arzate Cruz and Takeo Igarashi. A survey on interactive reinforcement learning: Design principles and open challenges. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, pp. 1195–1209, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369749. doi: 10.1145/3357236.3395525. URL https://doi.org/10.1145/3357236.3395525.

Serena Booth, W. Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(5):5920–5929, Jun. 2023. doi: 10.1609/aaai.v37i5.25733. URL https://ojs.aaai.org/index.php/AAAI/article/view/25733.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2023. URL http://github.com/Farama-Foundation/Gymnasium-Robotics.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, Feb 2021. ISSN 1476-4687. doi: 10.1038/s41586-020-03157-9. URL https://doi.org/10.1038/s41586-020-03157-9.

Benjamin Eysenbach, Sergey Levine, and Ruslan Salakhutdinov. Replacing rewards with examples: Example-based policy search via recursive classification. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=UVQNdLIELSU.

Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/c9319967c038f9b923068dabdf60cfe3-Paper.pdf.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1582–1591. PMLR, 2018. URL http://proceedings.mlr.press/v80/fujimoto18a.html.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1025–1037. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/gupta20a.html.

Abhishek Gupta, Aldo Pacchiano, Yuexiang Zhai, Sham Kakade, and Sergey Levine. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 15281–15295. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6255f22349da5f2126dfc0b007075450-Paper-Conference.pdf.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Deep Reinforcment Learning Symposium*, 2017.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2024. URL http://github.com/google/flax.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2), apr 2017. ISSN 0360-0300. doi: 10.1145/3054912. URL https://doi.org/10.1145/3054912.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022. ISSN 2504-4990. doi: 10.3390/make4010009. URL https://www.mdpi.com/2504-4990/4/1/9.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 10026–10039, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/531db99cb00833bcd414459069dc7387-Abstract.html.

Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. A survey of reinforcement learning from human feedback, 2024.

Ilya Kostrikov. JAXRL: Implementations of Reinforcement Learning algorithms in JAX, 10 2022. URL https://github.com/ikostrikov/jaxrl2. v2.

Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning.* PhD thesis, University of Illinois at Urbana-Champaign, USA, 2004. AAI3130966.

Jianlan Luo, Perry Dong, Yuexiang Zhai, Yi Ma, and Sergey Levine. RLIF: Interactive imitation learning as reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oLLZhbBSOU.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training, 2023.

Farzan Memarian, Zhe Xu, Bo Wu, Min Wen, and Ufuk Topcu. Active task-inference-guided deep inverse reinforcement learning. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 1932–1938, 2020. doi: 10.1109/CDC42340.2020.9304190.

Anis Najar and Mohamed Chetouani. Reinforcement learning with human advice: A survey. *Frontiers in Robotics and AI*, 8, 2021. ISSN 2296-9144. doi: 10.3389/frobt.2021.584075. URL https://www.frontiersin.org/articles/10.3389/frobt.2021.584075.

Cyrus Neary, Christos Verginis, Murat Cubuktepe, and Ufuk Topcu. Verifiable and compositional reinforcement learning systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pp. 615–623, 2022.

Cyrus Neary, Christian Ellis, Aryaman Singh Samyal, Craig Lennon, and Ufuk Topcu. A multifidelity sim-to-real pipeline for verifiable and compositional reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4349–4355, 2024. doi: 10.1109/ICRA57147.2024.10610735.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pp. 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606122.

Xinlei Pan and Yilin Shen. Human-interactive subgoal supervision for efficient inverse reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pp. 1380–1387, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Comput. Surv.*, 54(5), jun 2021. ISSN 0360-0300. doi: 10.1145/3453160. URL https://doi.org/10.1145/3453160.

Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(Volume 3, 2020):297–330, 2020. ISSN 2573-5144. doi: https://doi.org/10.1146/annurev-control-100819-063206. URL https://www.annualreviews.org/content/journals/10.1146/annurev-control-100819-063206.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/ross11a.html.

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering, 2019.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 9460–9471. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/3d719fee332caa23d5038b8a90e81796-Paper-Conference.pdf.

Henry Sowerby, Zhiyuan Zhou, and Michael L. Littman. Designing rewards for fast learning, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *J. Artif. Int. Res.*, 73, May 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12440. URL https://doi.org/10.1613/jair.1.12440.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.

Marcel Torne Villasevil, Max Balsells I Pamies, Zihan Wang, Samedh Desai, Tao Chen, Pulkit Agrawal, and Abhishek Gupta. Breadcrumbs to the goal: Supervised goal selection from human-in-the-loop feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=uOEeui0rL7.

Maryam Zare, Parham M. Kebria, Abbas Khosravi, and Saeid Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges, 2023.

Zichen Zhang, Yunshuang Li, Osbert Bastani, Abhishek Gupta, Dinesh Jayaraman, Yecheng Jason Ma, and Luca Weihs. Universal visual decomposer: Long-horizon manipulation made easy. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6973–6980. IEEE, 2024.

Xinglin Zhou, Yifu Yuan, Shaofu Yang, and Jianye Hao. Mentor: Guiding hierarchical reinforcement learning with human feedback and dynamic distance constraint, 2024.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

A Edge selection

The `selectEdge` subroutine governs how ILG-Learn allocates learning intervals while exploring the ILG. As described in Section 4, ILG-Learn starts at the source of the ILG and iteratively extends a tree of best-known policies to the other vertices of the ILG. The key insights that inform the design of `selectEdge` are:

- Training an edge policy $\pi_{(u,v)}$ can only begin once the policy π_u is learned (frozen), so it makes sense to stop investing learning intervals along an edge once a desired success probability **successThreshold** is reached. This way, future learning intervals can be allocated to downstream edges.
- Different edges of the ILG may require vastly different numbers of training episodes to acquire a successful policy, so it makes sense to dovetail learning intervals between edges so we do not spend too much effort on a particularly difficult edge.
- Only one feasible path needs to be found, so it makes sense to use a best-first search heuristic to avoid exploring the entire ILG.

Our edge selection algorithm is inspired by the exploration order found in Dijkstra's algorithm. In fact, if we set the parameter **intervalsLimit** to 1, our `selectEdge` subroutine implements an exploration order suitable for Dijkstra's algorithm. When **intervalsLimit** is greater than 1, `selectEdge` incorporates dovetailed exploration with early stopping heuristics to try to reduce the number of training episodes needed to learn a satisfactory policy.

Appendix A.1 describes the high-level structure of `selectEdge` that implements dovetailed exploration and early stopping. Appendix A.2 details the heuristic scoring function that `selectEdge` uses to implement best-first exploration. Appendix B provides advice for selection of relevant parameters. The choice of parameters used for the experiments in Section 5 are detailed in Table 2.

A.1 Dovetailed exploration

To implement early stopping and dovetailed exploration, ILG-Learn keeps track of these vertex and edge sets:

- *exploredVertices*. These are the vertices to which ILG-Learn has learned and solidified a path policy.
- *learnedEdges*. These are edges for which we have fully learned *and frozen* a policy. We freeze a policy once either (1) the empirical success probability exceeds **successThreshold** or (2) **intervalsLimit** learning intervals have been allocated to the edge.
- *abandonedEdges*. These are edges that we will never invest learning effort into because we have already found a better path to get to all of their successors.
- *frontierEdges*. These are edges leaving the explored tree (that is, leaving *exploredVertices*) that are neither fully "learned" nor "abandoned". At each iteration, we need to choose one edge from this set to invest a learning interval into.

All edge sets are initially empty. The logic for maintaining these sets is shown in Algorithm 2. To reduce notational clutter, we assume that *frontierEdges*, and *learnedEdges*, *abandonedEdges* are global variables that persists across calls to `selectEdge`. We also assume global access to *reachProb* (updated in line 15 of Algorithm 1) as well as the existence of the bookkeeping functions:

- $bestSuccessProb(u, v)$ returns the highest estimated probability found by line 13 of Algorithm 1 during any learning interval allocated to (u, v).
- $intervalsElapsed(u, v)$ returns the number of learning intervals that have been allocated to (u, v)

Algorithm 2: `selectEdge`

```
Input:
  • ILG structure $(U, E, u_0)$
  • Parameters intervalsLimit and successThreshold (introduced in Section 4)
  • Scoring heuristic parameters extensionPenalty and exploitationBonus (introduced in
    Appendix A.2)
  • Global access to reachProb (maintained by Algorithm 1)
Output: Edge $(u, v)$ for next learning interval, or $None$ if learning is complete.
$learnedEdges \leftarrow learnedEdges \cup \{(u,v) \in E \mid intervalsElapsed(u,v) \geq \mathbf{intervalsLimit}\}$;
$learnedEdges \leftarrow learnedEdges \cup \{(u,v) \in E \mid \vee\ bestSuccessProb(u,v) > \mathbf{successThreshold}\}$;
$abandonedEdges \leftarrow abandonedEdges \cup \{(u,v) \mid \forall u', u \prec u' \rightarrow reachProb_u < reachProb_{u'}\}$;
$exploredVertices \leftarrow \{u_0\} \cup \{v \in U \mid (\exists u, (u,v) \in learnedEdges) \wedge (\forall u, (u,v) \in E \rightarrow (u,v) \in$
  $learnedEdges \cup abandonedEdges)\}$;
$frontierEdges \leftarrow \mathrm{outgoingEdges}(exploredVertices) \setminus learnedEdges \setminus abandonedEdges$;
return $argmax_{(u,v) \in frontierEdges}\,\mathtt{score}((u,v))$
```

A.2 Score

The `selectEdge` subroutine (Algorithm 2) uses a heuristic `score` function to implement best-first exploration. We design `score` to balance the following desiderata:

- **Exploitation.** Perform best-first search by extending paths that have low cost.
- **Even exploration.** Dovetail exploration of multiple edges, trying to assign the same amount of training intervals to all extensions of the "sufficiently low cost" paths.
- **Anticipation.** Prefer extending paths that have few edges remaining to a final vertex.

As in Appendix A.1, we will assume global access to *reachProb* and access to the bookkeeping function *intervalsElapsed.* The score of an edge comprises three terms:

$$\begin{aligned}\texttt{score}(u, v) =& \, \mathit{exploitationIncentive}(u, v)\\ &- \mathit{intervalsElapsed}(u, v)\\ &- \mathit{minNumberOfEdgesToAFinalVertex}(v) \cdot \textbf{edgeExtensionPenalty}\end{aligned}$$

We describe the components as follows:

Term 1: Exploitation. The exploitation incentive strongly prioritizes investing training effort in edges that *could be part of the highest probability extension of some path in the tree explored so far.* Concretely, let

$$\mathit{bestExtensionProb} = \max_{(u,v) \in \mathit{frontierEdges}} \mathit{reachProb}(v)$$

Now let

$$\mathit{exploitationIncentive}(u, v) = \begin{cases} \textbf{exploitationBonus} & \mathit{reachProb}(u) \geq \mathit{bestExtensionProb} \\ 0 & \text{otherwise} \end{cases}$$

wher **exploitationBonus** is some very large number.

Term 2: Even exploration. Subtracting *intervalsElapsed*(u, v) softens the best-first search by trying to evenly allocate intervals to promising edges. This term will be much less in magnitude than **exploitationBonus**, so it acts as a tie-breaker within our soft best-first search.

Term 3: Anticipation. A high choice of **edgeExtensionPenalty** strongly strongly prioritizes extending paths that are close to reaching a final vertex (in terms of how many edges there are).

B Parameter selection

We now provide advice for how to select the parameters introduced in Section 4:

- **illustrationCount**: This should be enough goals so that the underlying example-based control algorithm can quickly learn a good policy. In our 2D navigation tasks, we found 10 goal examples to be sufficient, while for our robotic manipulation environments we found significantly better performance with 50 examples than 10. The choice of **illustrationCount** is tied to the specifics of the underlying example-based control algorithm; we believe that ILGs that include landmarks that admit a diverse set of success observations (e.g. a robotic arm can successfully grasp an object with a wide variety of arm angles and grip positions) will benefit from a relatively large number of success examples.

- **episodeLength**: The fixed time horizon used when training each edge policy must be long enough to allow the example-based control algorithm to explore the environment. The ideal value is usually significantly more than the number of timesteps needed for an expert to complete any given edge task.
- **intervalLength** and **intervalsLimit**: The quantity **intervalLength** × **intervalsLimit** should be enough training episodes for example-based control to saturate the success probability of the edge policy for any edge in the ILG. Since the number of learning episodes needed by example based control varies greatly depending on random seed, we recommend over-estimating this quantity. To avoid wasting training episodes on already-good policies, we recommend choosing a relatively low value for **intervalLength**: Since the choice **intervalLength** governs the frequency of teacher-intervention (in the form of responses to success/failure queries), we recommend choosing **intervalLength** to be large enough that there is only a modest amount of teacher interaction, yet low enough that learning episodes are not wasted.
- **successThreshold**: We recommend choosing a value of **successThreshold** slightly lower than the success rate of an optimal policy. Choosing a high **successThreshold** causes ILG-Learn to spend a large amount of learning effort on an edge before exploring deeper in the ILG, which may increase the success rate of the final policy at the cost of more environmental and student-teacher interactions.
- **estimationQueries**: We recommend choosing a number that gives adequate confidence in the success rate of the learned policy. In practice, we have observed that example based control often yields policies that either succeed much more or much less than our desired **successThreshold**, so relatively few *estimationQueries* (e.g. 30) suffice. If there are multiple feasible tasks in the ILG, choosing a high value of **estimationQueries** will improve ILG-Learn's ability to perform best-first search. We chose not to implement a statistically-rigorous version `estimateProbability` since doing so would in general require the teacher to respond to many success//failure queries and was not necessary for good end-to-end performance in practice.

as well as the additional parameters introduced in Appendix A.2:

- **exploitationBonus**: We recommend choosing a very high value, so that `score` function implements a "soft" best first seach, in which the second and third terms of the `score` function serve as tie-breakers. To achieve this, one can choose parameters such that **exploitationBonus** > **intervalsLimit** + $\mathit{diameter}(G)$ × **edgeExtensionPenalty** where $\mathit{diameter}(G)$ is the diameter of the ILG.
- **edgeExtensionPenalty**: We recommend choosing **edgeExtensionPenalty** to be a guess of how many learning intervals will be required to train an edge along a feasible path to reach the **successThreshold.** Since choosing this parameter requires considerable foresight, we recommend choosing a relatively low value.

C Implementation details

C.1 ILG-Learn and RCE

We implement ILG-Learn in Python. We re-implemented RCE (following instructions of Eysenbach et al. (2021)) on top of JaxRL2's (Kostrikov, 2022) implementation of SAC (Haarnoja et al., 2017). We use this RCE implementation both as the example-based control subroutine in our ILG-Learn algorithm and as the RCE baseline.

Hyperparameters. We use the same SAC+RCE hyperparameters as Eysenbach et al. (2021), including the SAC-specific hyperparameters that were inherited from Haarnoja et al. (2017), although we increased the width of each hidden layer in the actor and critic MLPs from 128 to 256. The original RCE implementation varied the "n-step returns" and "Q combinator" hyperparameters depending on the particular task. We use 10-step returns for all tasks, and use *min* as the "Q combinator" in our `Stack` family of environments (this is the typical choice of Q combinator, following Fujimoto et al. (2018)). We found that the *max* combinator works better in the `Maze` environments and use *max* for those environments.

Our choice of ILG-Learn-specific parameters (detailed in Appendix B) is shown in Table 2.

	`DiagonalMaze3x3`	`DiagonalMaze5x5`	`DiagonalMaze7x7`	`DiagonalMaze7x7-Coarse`	`Maze4x4-Fine`	`Maze4x4-Coarse`	`StackAB`	`StackChoice-Outwardview`	`StackBA-Outwardview`
illustrationCount	10	10	10	10	10	10	50	50	50
episodeLength	400	400	400	1200	200	600	80	80	80
intervalLength	100k	100k	100k	100k	100k	100k	100k	100k	100k
intervalsLimit	100	100	100	100	100	100	100	100	100
successThreshold	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
estimationQueries	30	30	30	30	30	30	30	30	30
exploitationBonus	101	101	101	101	101	101	101	101	101
edgeExtensionPenalty	3	3	3	3	3	3	3	3	3

Table 2: ILG-Learn-specific parameter selection.

Policy sequencing details. Underlying ILG-Learn's compositional approach to long-horizon policy learning is the `sequencePolicies` subroutine that sequences edge policies to form a path policy. As described in Section 4, each edge policy is executed for a fixed horizon (governed by the **episodeLength** parameter). To formalize the behavior of `sequencePolicies` it is convenient to associate each (path or edge) policy π with a *horizon* denoted $\mathit{horizon}_\pi$). We assume that the array *horizon* of such bookkeeping variables is in global scope and define the `sequencePolicies` subroutine in Algorithm 3.

Algorithm 3: `sequencePolicies`

```
Input:
    • Path policy $\pi_u$ and edge policy $\pi_{(u,v)}$
    • Parameter episodeHorizon
    • Mutable bookkeeping dict horizon
Output: Path policy to reach $v$
Function $\pi_v(\tau)$:
    if $|\tau| \leq \mathit{horizon}_{\pi_u}$ then
        return $\pi_u(\tau)$;
    else
        return $\pi_{(u,v)}(\tau)$;
$\mathit{horizon}_{\pi_v} \leftarrow \mathit{horizon}_{\pi_{(u)}} + \mathit{horizon}_{\pi_{(u,v)}}$;
return $\pi_v$
```

If *horizon* is maintained such that $\mathit{horizon}_{\pi_{(u,v)}}$ is **episodeLength** for each edge policy $\pi_{(u,v)}$ then Algorithm 3 matches the textual description of Section 4. In our implementation, we noticed that running each edge policy for the fixed horizon resulted in non-smooth motion, because the agent would hesitate after having achieved each landmark (waiting for the associated edge policy's horizon to be fully consumed) before moving on to the next. To reduce this "waiting time" we slightly enriched the information provided by the teacher: every time the learner uses `querySuccess`(v) at the end of an episode to assess the success rate of an edge policy $\pi_{(u,v)}$, if the teacher responds "Success" they also provide the index of timestep at which the agent entered βv. Then we update $\mathit{horizon}_{\pi_{(u,v)}}$ to be the max of these success timestep indices, plus an addition 15 timesteps of slack term. The results in Section 5 were collected using the above procedure, although we do not believe that the details of our `sequencePolicy` heuristic is an important aspect of ILG-Learn.

Success estimation details. As described in Section 4, the `estimateProbability` estimates the probability that a learned edge policy $\pi_{(u,v)}$ successfully reaches βv. Since we start each rollout of $\pi_{(u,v)}$ from the state distribution induced by executing the path policy π_u, we naturally obtain an empirical estimate of the probability that the path policy obtained by sequencing π_u and $\pi_{(u,v)}$. We call attention to this because our "Dijkstra-style" planning implemented by `selectEdge` only receives the costs of *paths*, not individual edges. If we assume that each edge has an intrinsic fixed cost, then this is an inconsequential bookkeeping detail. However, since $\pi_{(u,v)}$ could reach $\beta(v)$ even if not started from $\beta(u)$ (and moreover because our cost estimates are empirical) it is possible for the estimated cost of a path to be *less* than that of one of its prefixes. Such ill-behaved path probabilities only pose a problem for ILG-Learn if a high-cost prefix dissuades `selectEdge` from exploring what would become the lowest-cost source to sink path. Such a situation is only likely to arise if the teacher provides an ILG that contains landmarks that are both (1) hard to reach and (2) not necessary to scaffold exploration towards subsequent landmarks.

C.2 BC baseline

We implemented the BC baseline in Python using Jax (Bradbury et al., 2018) and Flax (Heek et al., 2024). For each task, we pooled the $(\text{observation}, \text{next action})$ pairs from all the demonstrations and trained an MLP to predict the next action given the current observation. Our MLP had 2 hidden layers (each of width 512), we used the Huber loss and a learning rate of 0.0001. For each task, we had a heldout set of validation demonstrations. We stopped training when the validation loss stopped decreasing.

D Environment Details

Our manipulation environment is built in `robosuite` (Zhu et al., 2020). Our maze environment is built in de Lazcano et al. (2023) and is a customized of the Maze2D environment originally introduced in D4RL (Fu et al., 2020). All our experiments use Gymnasium Towers et al. (2023) and MuJoCo Todorov et al. (2012).

D.1 Stack

We adapt the "Stack" environment that is included in `robosuite`. This environment simulates a 7-DoF Franka Panda robot arm. We use the provided 7-dimensional continuous action space that includes an operational space controller with fixed impedance. We also use the provided observation space, which is 55-dimensional and comprises the robot's joint angles, joint velocities, end effector position, end effector quaternion, gripper position, gripper velocity, the position of each block, the quaternion of each block, the gripper-object distance of each block, and the distance between the two blocks.

We make the following modfications to the original environment:

1. The side length of each cubical block is reduced from 0.05 meters to 0.04m. We do this so we can include more randomness in the initial block placements.

2. The two blocks always initialize along the centerline (parallel to the x-axis in the simulation, which is *depth* from the perspective of the robot) of the table with block A closer to the robot than block B. The exact positions are determined randomly in each reset as follows: we uniformly select two x coordinate values in the range $(-0.2, 0.2)$. If the values are at least 0.05 meters apart, we let block A start at the lower x value and block B start at the greater x value. Otherwise, we repeat this process until we obtain sufficiently spaced x values.

3. For **StackChoice-Outwardview** we allow the blocks to be stacked in either order; for **StackBA-Outwardview** we require block B to be stacked on block A.

4. For **StackChoice-Outwardview** and **StackBA-Outwardview** we implement simulated occlusion, as if the observations are produced by an object detector that is mounted on the same side of the table as the robot. To do this, we position a MuJoCo camera at the position indicated by the pink camera icon in Figure 1. At each timestep, if *no pixel* belonging to either block A or block B

is present in the ground-truth object segmentation yielded by this camera we mask all components of the observation corresponding to that cube with the value -2 (which is far from the observation values encountered in normal operation).

To generate the landmark examples and demonstrations for behavior cloning, we use a handwritten scripted policy. We inject noise into this scripted policy so that it covers a variety of grip locations and arm paths. We also tried using teleoperated demonstrations (and examples gathered therefrom) for the `StackAB` task and did not find significantly different performance so we elected to use scripted policies to generate all data. We include 40 demonstrations, along with an additional 10 held-out validation demonstrations to evaluate loss during BC training.

The example observations for the "grasp" landmarks are taken after the arm has grasped *and lifted the block by* $\sim$0.02 *meters*. We found that slightly lifting the block greatly improves RCE's ability to learn a successful policy, even in the absence of occlusion. We believe this is because grasps that do not lift the block are very close in observation space to "near grasps" that do not make firm contact with the block, yielding a difficult discrimination task for the RCE critic.

To simulate the teacher's response to the `querySuccess` queries during training, we defined $\beta(grasp\ A)$ to be all states where block A is at least 0.021 meters above the surface of the table (and similarly for $\beta(grasp\ B)$. We defined $\beta(stack)$ analogously to the success condition of the original `robosuite` stacking environment: the cubes must be touching each other, the gripper cannot be touching the block on top of the tower, and the block on top must be both above the surface of the table and aligned horizontally with the other block.

D.2 Maze

We customized Gymnasium Robotics' Maze2D environment; our layouts are shown in Figure 4. The agent is a force-actuated point mass with a 2-dimensional continuous action space. The observation is continuous and 4-dimensional, comprising the agent's current position and velocity, but not the goal location or the location of any of the maze's walls.

Each landmark is the center of a room, as illustrated in Figure 4. Each square room has a side-length of 3.6; the landmark region is a circle of diameter 1 centered within the room. Only the agent's position (not velocity) determines membership in a landmark. To provide illustrative observations, we include one illustrative observation from the center of the landmark, with the remaining illustrative observations drawn uniformly at random from positions within the landmark. All illustrative observations contain 0 velocity.

The demonstrations are gathered with handwritten scripted policies that include a small amount of noise to better cover the state space. For each task, all demonstrations traverse the same sequence of rooms to reach the final goal. For each environment, we include 10 demonstrations and an additional 5 held-out demonstrations for evaluating loss during training.

E Learning curves

Learning curves are found in Figure 7. We report the overall success rate of the entire task; for ILG-Learn this means that the learner must explore a source-to-sink path before achieving nonzero success rate. Note that some trials of ILG-Learn stop before reaching 10m environment steps, because ILG-Learn allows early stopping. In this case, we extend the success rate of the final policy for the rest of the 10m steps (even though no further training occurs).

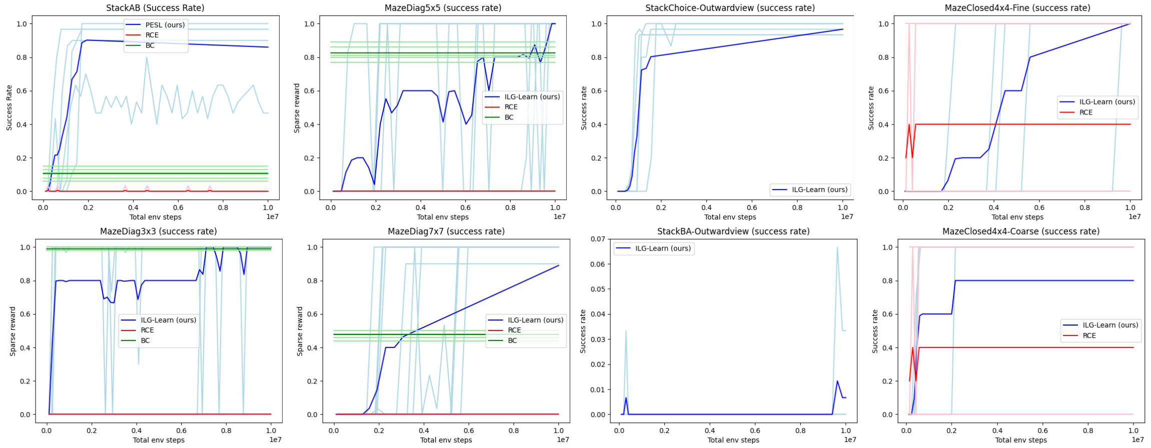


Figure 6: Learning curves showing success rate over total environment steps (including environment steps used during resets and calls to `estimateSuccess`. The light lines show individual random seeds, the dark lines show the mean of 5 random seeds.

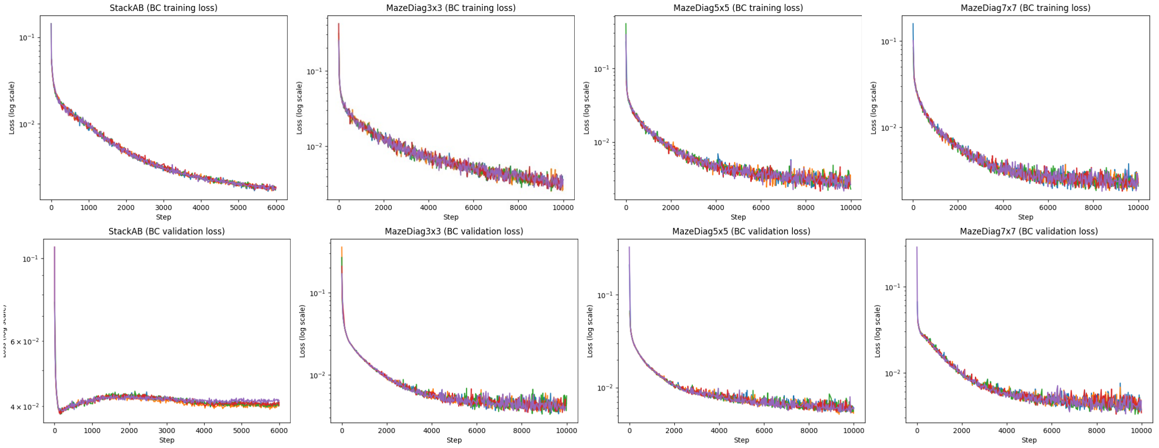


Figure 7: Learning curves showing Huber loss during offline BC training. Each line represents one of 5 random seeds. We chose a number of training steps that was sufficient for validation loss to stop decreasing. For `StackAB` we found this to occur at about 6000 steps, for the other environments we found this to occur at about 10000 steps.