# Extended Abstract

**Motivation** Humans frequently encounter environments where priorities must shift rapidly in response to new, uncertain information—such as a day trader reacting to volatile markets or an arcade-goer maximizing fun with limited resources. This project investigates whether Hierarchical Reinforcement Learning (HRL) agents can emulate such adaptive behavior in the face of hidden and shifting rewards. Specifically, the study explores if HRL agents can strategically select subgoals and recognize when to cut their losses (via a "recall" action) in a reward-uncertain setting, extending beyond prior works that focus on simpler recovery or adaptation mechanisms.

**Method & Implementation** A custom Gridworld environment was developed where an agent must collect items to reach a target inventory value of 100 as efficiently as possible. While past item values are known, current values remain hidden until an item of that type is picked up, introducing reward uncertainty. The values are generated randomly from 4 different distributions, some of which are inspired by leading stock market models. The agent can move in four directions or use a recall action to return to the start, forfeiting collected items.

The HRL architecture features an upper policy (rewarded for inventory progress) and a lower policy (rewarded for subgoal achievement and environment feedback). The most effective approach combined Behavioral Cloning (BC) with Conservative Q-learning (CQL), while other techniques such as Hindsight Experience Replay (HER) and Random Network Distillation (RND) did not yield improvements. Several grid layouts (Maze, No Walls, Cross, Unbalanced Cross) were used for training and evaluation. Comparisons included a base model without BC or CQL, and models where we disabled recalling and/or switched out our default Greedy Search expert for an optimal policy expert . Experiments were also conducted testing variations on upper policy implementations, which primarily focused on adjusting the extent to which the upper policy controlled the recall action.

**Results** Performance varied across environments and methods. In the Maze layout, the best HRL method (CQL-R) achieved a 10.6% success rate, outperforming the base agent (0%) but lagging far behind the Greedy expert (100%). In the No Walls layouts, HRL agents with more conservative recalling (CQL-R, RH + MR) succeeded over 80% of the time (compared to 0% for the based model), while those which attempted to recall more had trouble achieving the goal consistently. In the Cross and Unbalanced Cross layouts, non-base HRL agents approached expert-level success rates (100% or near), with improved average rewards and speed relative to the optimal path. Methods utilizing recalls more performed especially well, performing even better than the Greedy expert on Unbalanced Cross in both Rewards/Episode and Speed wrt. Optimal. Finally, our upper policy variations showed potential with higher low success rates, but tended to train slower. However, the upper policy variations showed superior results on the No Walls layout, suggesting that there is room for further exploration.

**Discussion & Conclusion** Our problem proved nontrivial for RL agents. Improvements to the observation space and rewards were likely needed, but time constraints limited us. We attempted an $RL^2$ meta-RL baseline, but couldn't finish implementation. Removing the upper policy might have yielded better results, though at the cost of performance in high-variance environments. Despite numerous bugs being introduced by our hacky Gymnasium environment, which hid several bugs until late, we believe our setup presents a valuable and challenging benchmark for adaptive agents, reflecting real-world adaptation tasks.

Future directions include expanding to continuous action spaces for realism (e.g., household robots), exploring suboptimality detection and human-in-the-loop learning, and refining our recall action, which complicated training but enables richer policy behavior.

# Exploration in a Reward Uncertain Environment

**Denis Liu**
Department of Computer Science
Stanford University
dfliu@stanford.edu

**Victor Li**
Department of Computer Science
Stanford University
vli42@stanford.edu

## Abstract

This study investigates the abilities of hierarchical reinforcement learning (HRL) agents to adapt to environments with hidden and shifting rewards through strategic subgoal selection and recall mechanisms. We develop a custom Gridworld environment where agents must collect items with uncertain values to reach target inventory scores. Our implementation combines behavioral cloning with conservative Q-learning in a hierarchical architecture, achieving 81.9% success in open layouts but revealing fundamental limitations in complex mazes (10.6% success vs 100% expert baseline). Ultimately, we find that recall actions, while theoretically valuable, destabilize training; CQL improves learning under sparse feedback; and hierarchical policies struggle with continuously shifting rewards.

## 1 Introduction

Modern decision-making agents must navigate environments where reward structures shift unexpectedly–from financial traders reacting to volatile markets to robotic systems adapting to equipment failures. While humans demonstrate remarkable capacity for strategic course-correction, existing reinforcement learning methods struggle with these dynamics. Our work attempts to address this gap through two key contributions: (1) A novel Gridworld environment modeling hidden reward discovery and strategic recall and (2) An HRL architecture combining behavioral cloning with conservative Q-learning.

## 2 Related Work

Adaptive reinforcement learning in non-stationary and partially observable environments has attracted increasing interest due to its relevance in real-world settings such as robotics, autonomous navigation, and personalized recommendation systems. Our work is inspired by three key areas: strategic adaptation to reward uncertainty, hierarchical control for long-horizon decision making, and memory-augmented RL for effective recall of previously observed information.

### 2.1 Adaptation

Several prior approaches explore how agents can adapt to dynamically shifting rewards. Du et al. (2024) , for instance, propose a trial-and-error strategy for rapid online adjustment by detecting suboptimality at deployment time. However, they address suboptimality by utilizing a simple fallback policy (an arm retraction) and lack a mechanism to address a task which requires more complex and strategic abandonment.

Meta-RL approaches such as Duan et al. (2016) and Finn et al. (2017) enable fast task adaptation via gradient-based meta-training, but typically assume clear episodic task boundaries (i.e., a distinction between meta-exploration and meta-testing phases)—an assumption that does not hold

in our Gridworld environment. In our setting, task identity is latent and may shift mid-episode, effectively blurring the distinction between meta-exploration and meta-testing. Further improvements to meta-RL, such as the decoupling strategy proposed by Liu et al. (2020), enhance generalization by separating exploration and exploitation strategies. However, this strategy relies on explicit task identifiers during training for conditioning or relabeling, a luxury not easily available in our environment with hidden and dynamic reward structures.

### 2.2 Hierarchical Reinforcement Learning (HRL)

HRL introduces temporal abstraction through subgoals or skills, enabling agents to handle long-horizon tasks with sparse feedback. FeUdal Networks (FuNs) Vezhnevets et al. (2017) and HIRO Nachum et al. (2018) are among the seminal architectures in this domain. FuNs utilize a manager-worker hierarchy with explicit goal setting, whereas HIRO improves sample efficiency through subgoal relabeling and off-policy learning. Our architecture draws inspiration from HIRO but eschews the relabeling strategy as it seems unsuited for discrete action spaces.

### 2.3 Behavioral Cloning and Offline Objectives

Finally, our work leverages behavior cloning (BC) to initialize subgoal policies and conservative Q-learning (CQL) (Kumar et al., 2020) to stabilize value estimation under partial observability. CQL has been shown to improve robustness in offline settings by avoiding value overestimation on out-of-distribution actions, particularly in sparse reward contexts. We also attempt to integrate strategies such as hindsight experience replay (HER) (Andrychowicz et al., 2017) as well as random network distillation (RND) (Burda et al., 2018).

## 3 Method

Our approach involves (1) environment creation and (2) experimenting with hierarchical RL to solve the problem.

### 3.1 Environment

Our environment problem is defined as follows:

For each episode, the agent starts at some position (which may be randomized or fixed depending on the map). There are items of 5 different types. Each item type possesses a "history" of previous values, as well as a hidden (to the agent) current value. These values are assigned by randomly generating them on reset by sampling from one of four random distributions with the intention to create an artificial stock market of sorts.

- **Pure Normal Distribution**
  Here, the current value and every value in the history are all sampled from the same normal distribution.
- **Progressive Normal Distribution**
  Here, the each value is sampled from a normal distribution that uses a preset $\sigma$ (standard deviation) and $\mu$ (mean) equal to the value sampled right before it in the history.
- **Geometric Brownian Motion (Maiti, 2021)**
  This is commonly used to model stock prices. We use the Euler–Maruyama method (with a time step of 1.0) to generate new samples. Each sample is defined as

$$S_t = S_{t-1} \cdot \exp\left(\mu - \frac{1}{2}\sigma^2 + \sigma \cdot \varepsilon\right)$$

  where $\mu$ is drift, $\sigma$ is volatility, and $\varepsilon$ randomly generated noise sampled as $\varepsilon \sim \mathcal{N}(0, 1)$
- **Jump Diffusion**
  We simulate a discrete-time approximation of Merton's Jump Diffusion Model (Merton, 1976), which is an extension of Geometric Brownian Motion that includes large, random jumps. Each sample is defined as

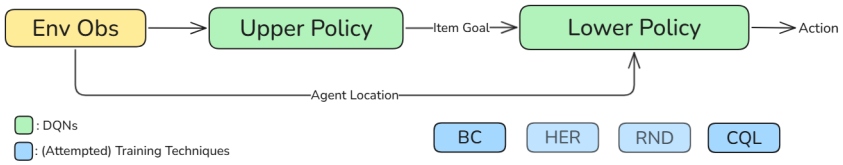



Figure 1: Hierarchical decision-making framework. The upper policy $\pi_u$ issues subgoals $g_t \in \mathcal{I}_{loc} \cup$ recall based on global state $s_t$. The lower policy $\pi_l$ executes primitive actions $a_t \in \mathcal{A}$ conditioned on $g_t$. Dashed arrows indicate reward signal flow.

$$\log\left(\frac{S_t}{S_{t-1}}\right) = \mu - \frac{1}{2}\sigma^2 + \sigma \cdot \varepsilon + \sum_{i=1}^{N} Y_i$$

$\mu$, $\sigma$, and $\varepsilon$ are the same as in Geometric Brownian Motion, $N$ is a randomly generated number of jumps, and $Y_i$ are each randomly generated jump sizes for each jump.

At each step, the agent can take the typical grid actions (up, down, left, right) as well as a "recall" action. The agent picks items up when entering the square which they are located in, attaining a score equal to the value of the item. Because the agent is aware of its total inventory score, it also knows the exact current value of every other item of the same type (e.g., picking up a banana item for the first time reveals the value of all bananas on the grid). The objective is to obtain a specified total goal score in as few steps as possible.

"Recalling" is an action whereby the agent's position is reset to the initial position. The agent forfeits all of its collected items, which return to their original locations. Recalling does *not* reset the step counter, but the agent retains all knowledge it gleaned from its exploratory actions (i.e., item values). Recalling is intended to be used when the agent learns that an item value may not have been what it expected, unveiling a suboptimal initial trajectory. With a recall, the agent may find itself better positioned to take advantage of the new information. One of the primary purposes of our project was to improve agent recall usage, as it would show that the agent acquired some sense of an ability to adapt to surprising observations.

We assign the following environment **rewards** for each step:

- -1 if a recall is performed
- +1 if an undiscovered item is collected (to promote exploration)
- +0.1 if a discovered item is collected
- +10 if the goal score is reached
- -0.5 otherwise

### 3.2 Hierarchical Architecture

Denote the set of item locations to be $\mathcal{I}_{loc}$, the action space as $\mathcal{A}$, the observation space as $\mathcal{O}$, and the environment reward function as $\mathcal{R}$. Also, let $\tau_{i,j} := \langle s_i s_{i+1} \dots s_j, a_i a_{i+1} \dots a_j, r_i r_{i+1} \dots r_j, s_{i+1} s_{i+2} \dots s_{j+1}, g_{i,j} \rangle$ (just a subtrajectory with consistent goal $g_{i,j}$ starting at step $i$ and ending at step $j$). Finally, define $gs(\tau_{i,j}) \in [0, 1]$ to be the difference in inventory value at $s_j$ and $s_i$ all over the goal score. The function $gs$ represents the progress made toward the goal score for some subtrajectory. Our architecture decomposes decision-making into strategic and tactical levels (Fig. 1):

- **Upper Policy** ($\pi_u$**:** $\mathcal{O} \to \mathcal{I}_{loc}$):
  The upper policy takes in a full observation and selects a subgoal from item locations $\mathcal{I}_{loc}$. Given a subtrajectory $\tau_{i,j}$, the upper policy receives reward

$$R_u(\tau_{i,j}) = \sum_{k=i}^{j} r_k + 0.1 \cdot gs(\tau_{i,j}). \tag{1}$$

- **Lower Policy ($\pi_l$: $\mathcal{O}_{agent\_loc} \times \mathcal{I}_{loc} \rightarrow \mathcal{A}$)**:
  The lower policy takes in the agent state and a subgoal $g \in \mathcal{I}_{loc}$ from the upper policy and returns an action. The lower policy's reward function is:

$$R_l(s_t, g_t) = \mathbb{I}_{s_t=g_t} + 0.1 \cdot \mathcal{R}(s_t) \tag{2}$$

  where $\mathbb{I}_{s_t=g_t}$ is the indicator function.

**Key Implementation Details**:

- State/observation representation includes agent position, item locations, item values (if known), item histories, and whether the item has been picked up or not.
- The upper policy is called whenever the lower policy completes the subgoal or if $n$ amount of timesteps are reached. In our experiments, we set $n = 30$.
- Both the upper and lower policies are modeled as DQNs.

### 3.3 Training Techniques

We implemented several RL enhancements. Namely, we tried different levels of permuting Hindsight Experience Replay (HER), Random Network Distillation (RND), Behavioral Cloning (BC), and Conservative Q-learning (CQL).

In particular, HER was implemented while collecting trajectories for the lower policy by relabeling the goal provided by the upper policy with actual item locations that the agent reached along the way. Similarly, we implemented RND into our Q-networks because we found that the agent would find itself going back and forth between two states frequently. After trying several variations of RND (both in the higher and lower policies) and HER, we found that the methods provided no significant improvement in initial experiments, and thus, leave the results out of this paper.

Because model convergence tended to be extremely slow without BC, we implemented BC and explored its effects in our experiments. We experiment by providing different expert trajectories: those taken from a greed search policy which always paths efficiently to the closest item and an optimal policy which has all item value information and always selects the fastest path to achieve the desired goal score (implemented with a brute-force BFS).

Finally, we found that CQL, when used in conjunction with BC, typically led to quicker convergence, so we ultimately decided to stick to the CQL+BC combination for our additional experiments. We hypothesize that CQL improves performance because it helps the agent better align with the expert that BC was performed on.

### 3.4 Model Variations

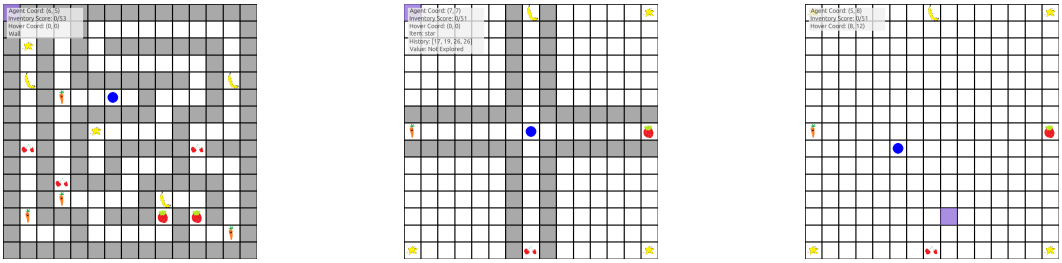
We also experimented with different variations of the upper policy, where we attempted to improve our model by granting the upper policy differing levels of control of the "recall" action. Our hope was to show that delineating the "recall" decision-making process to the upper policy would both lessen the learning load on the lower policy and enable more intelligent usage of the action because we found that the agent typically tended to only use recall to a detrimental effect.

Concretely, our variations to the upper policy are enumerated below:

- Modifying Reward: we removed environment rewards from the upper policy's reward function and used only progress as a metric. We do this because it encourages the upper policy to select a subgoal that is farther away, which would be negatively penalized by environment reward. That is,

$$R_u(\tau_{i,j}) = gs(\tau_{i,j}).$$

- Adding Initial State as Item Location Subgoal: by explicitly allowing the upper policy to select the initial state as a subgoal, we encourage the lower policy to take the recall action when the upper policy selects the initial state subgoal.
- Disabling Recall in the Lower Policy: rather than setting a subgoal for the lower policy, the higher policy instead is in full control of the recall action, and the lower policy only takes grid actions.



(a) Maze (b) Cross (c) No Walls

Figure 2: Example grid layouts used in training and evaluation.

## 4 Experimental Setup

We ran experiments on four grid layouts, which are meant to test different capabilities of our agents. Figures 2 shows the sample grids used for training and evaluation. Note that **Cross** and **Unbalanced Cross** have the same layout but differing item values.

- **Maze**: Complex navigation requiring exploration. Items are relatively sparse, and walls provide a major challenge. Item values and initial position are randomized.
- **No Walls**: Pure value comparison and route planning. The lack of walls makes navigation much more simple, as do the static item locations. Item values and initial position are randomized.
- **Cross**: Balanced risk/reward paths. Four items all lay at an equal distance from the fixed starting point (the center of the cross). Agents must always always go through the center in order to navigate to new items. Only item values are randomized.
- **Unbalanced Cross**: Quick selection of and navigation to the most valuable item. The only difference between this layout and the **Cross** layout is the item values. One of the four items, chosen at random, always has value greater than or equal to the goal score, with the other three having little to no value. Like **Cross**, the initial position isn't randomized

First, we compare the effects of BC, CQL, and the existence of the recall action to the greedy search baseline as well as a model which uses none of the RL enhancements. Then, we performed similar comparisons for the variations to the upper policy.

We trained each model for a set number of epochs for each map, varying the number of epochs by map difficulty, and tested them on the same 160 randomly generated maps for each grid layout using the following evaluation metrics:

- **Success Rate**: Whether the agent succeeds in achieving the goal score
- **Low Success Rate**: Whether the agent succeeds in reaching the lower policy goal
- **Rewards per Episode**: The average rewards obtained by the agent each episode
- **Speed wrt. Optimal**: The average speed of each path compared to the optimal policy. A path that fails to reach the goal score has a zero speed by default, and one which reaches the goal in the same amount of time as the optimal has a speed of one. More concretely, given a set of episodes $E$,

$$\text{speed}(\pi) = \frac{\sum_{e \in E} \text{speed\_of\_path}(\pi(e))}{|E|}$$

where

$$\text{speed\_of\_path}(p) = \begin{cases} \frac{|optimal|}{|p|} & \text{if } p.score \geq goal\_score \\ 0 & \text{if } p.score < goal\_score \end{cases}$$

and $\pi(e)$ is the generated path $p$ for episode $e$.

Table 1: Maze

| Method | Success Rate | Low Success Rate | Rewards/Episode | Speed wrt. Optimal |
|---|---|---|---|---|
| Base | 0 | 0.338 | -81 | 0 |
| CQL | 0.063 | 0.606 | -77.3 | 0.018 |
| CQL* | 0.01 | 0.569 | -82.5 | 0.004 |
| CQL-R | **0.106** | 0.619 | **-68.1** | **0.031** |
| CQL*-R | 0.081 | 0.625 | -68.9 | 0.019 |
| IP | 0.0188 | 0.475 | -81.8281 | 0.0056 |
| IP + MR | 0.0188 | **0.7** | -83.0581 | 0.0035 |
| RH + MR | 0.075 | 0.6438 | -75.0987 | 0.0153 |
| Greedy | 1 | – | -4.23 | 0.854 |

# 5 Results

## 5.1 Quantitative Evaluation

Tables 1-4 show key performance metrics across different environments. The best non-greedy result in each columnn is bolded (except for success rates where multiple of our models achieved a 1.0 success rate). We use the following shorthands for method name:

1. Base: the base model without BC or CQL
2. CQL: the base CQL model with a greedy expert for BC (default model which we ablate further)
3. CQL*: CQL with an optimal expert for BC (not the default since the optimal expert has access to hidden item values)
4. CQL-R: CQL model with greedy BC expert, but removing the ability to recall
5. CQL*-R: CQL model with optimal BC expert, but removing the ability to recall
6. Upper Policy Variations (same parameters as CQL)
    - IP: Adding Initial Position to upper policy subgoals
    - IP + MR: Adding Initial Position to upper policy subgoals and modifying upper policy reward function
    - RH + MR: Migrating Recall action completely to the upper policy and modifying upper policy reward
7. Greedy: Greedy search (the same as the greedy expert policy)

Critically, the Base method performs much worse than other agents in most scenarios. We also observe generally improved success rates without the recall action, showing that recalling tends to be an issue for the lower policy. This is further supported by the fact that our upper policy variations tended to have better low success rates. Despite this, we found the upper policy variations to have little improvement in success rate or reward, with the exception of the **No Walls** layout. For example, RH + MR reaches a reward/episode almost equivalent to CQL-R, the best performing model, while still using recalls. We hypothesize that the RH + MR method has the highest potential out of all the methods but takes longer to train as the upper policy's task becomes much harder.

## 5.2 Qualitative Analysis

We observe a few key patterns in recalling and BC expert tradeoffs.

### 5.2.1 Recalling

Recall actions negatively correlated with success across most maps. Since recalling resets the current inventory, doing so too frequently prevents the agent from effectively progressing to the goal score. From manual observation, we see that our low policy will often successfully reach an item, get assigned a goal item that is further away, and recall once it sees that it isn't getting to the item quickly enough.

Table 2: No Walls

| Method | Success Rate | Low Success Rate | Rewards/Episode | Speed wrt. Optimal |
|---|---|---|---|---|
| Base | 0 | 0.138 | -85.5 | 0 |
| CQL | 0.406 | 0.95 | -58.8 | 0.153 |
| CQL* | 0.231 | 0.931 | -65 | 0.08 |
| CQL-R | 0.819 | 0.988 | **-32.7** | 0.365 |
| CQL*-R | 0.706 | 1 | -39.7 | 0.298 |
| IP | 0.3875 | 0.9937 | -59.66 | 0.1528 |
| IP + MR | 0.606 | 1 | -44.7813 | 0.288 |
| RH + MR | **0.8375** | 1 | -33.2606 | **0.3769** |
| Greedy | 1 | - | -2.07 | 0.897 |

Table 3: Cross

| Method | Success Rate | Low Success Rate | Rewards/Episode | Speed wrt. Optimal |
|---|---|---|---|---|
| Base | 0 | 0.95 | -78.9 | 0 |
| CQL | 0.219 | 1 | -69.5 | 0.101 |
| CQL* | 0.563 | 1 | -48.3 | 0.353 |
| CQL-R | 0.494 | 1 | -52 | 0.226 |
| CQL*-R | **0.869** | 1 | **-17.2** | **0.703** |
| IP | 0.6313 | 1 | -44.7 | 0.3794 |
| IP + MR | 0.0188 | 1 | -77.68 | 0.0075 |
| RH + MR | 0.0375 | 0.9875 | -71.54 | 0.025 |
| Greedy | 1 | – | -5.2 | 0.943 |

Much of our work was made in an attempt to resolve this issue. In the end, we were not able to completely resolve the issue. However, we formulated the **Unbalanced Cross** as a grid specifically designed to encourage recalling. Because of the way the grid is designed, a single item is enough to reach the goal score, and the other three items put together will never be enough. Consequently, there is no downside to recalling after picking up one of the low-value items, except for negative environment reward. However, there is upside–the agent saves time by teleporting to the initial position, getting closer to the other items (including the single high value item). Because of this, models that were recalling more tended to work better on **Unbalanced Cross**.

#### 5.2.2 BC Expert

There are notable tradoffs between using an optimal expert versus using a greedy expert for behavior cloning. When trained with BC using the greedy search as an expert, our upper level policy learns to select closer item goals just like the greedy search. This worked better on the **Maze** and **No Walls** layouts, where the agents who made more consistent progress by picking up close items achieved better results.

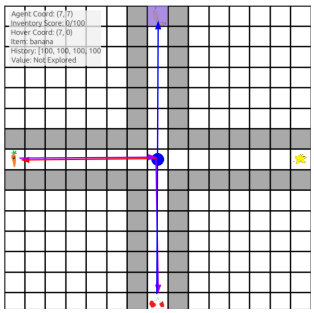
On the other hand, when trained with BC using the optimal policy as an expert, the higher level policy will learn to pick items with higher expected values at the potential cost of distance. On **Maze** and **No Walls**, the extra distance makes it more difficult for the lower level policy to reach each item goal, taking longer to do so even on successes. This also exacerbates the recalling problem discussed above. However, the equidistant items on **Cross** and **Unbalanced Cross** mitigate this downside, and the superior value-based selection allows the higher level policy to more consistently pick the high value items.

## 6 Discussion and Conclusion

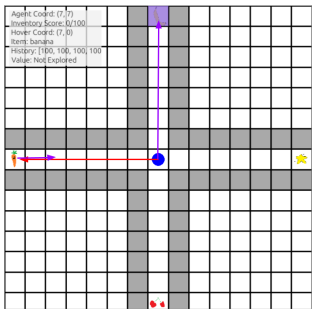
Ultimately, we found that our problem was nontrivial to solve for agents. Likely, several improvements could have been made to the observation space and reward functions, but we ran out of time. We also attempted to create an $RL^2$ agent as a meta-RL baseline, but didn't complete the implementation in time. In addition, we likely would have achieved closer-to-greedy results had we eschewed the upper

Table 4: Unbalanced Cross

| Method | Success Rate | Low Success Rate | Rewards/Episode | Speed wrt. Optimal |
|---|---|---|---|---|
| Base | 0.231 | 0.538 | -128 | 0.05 |
| CQL | 1 | 1 | -2.14 | 0.4 |
| CQL* | 1 | 1 | **1.81** | **0.474** |
| CQL-R | 1 | 1 | -1.86 | 0.369 |
| CQL*-R | 0.975 | 1 | -2.23 | 0.4 |
| IP | 1 | 1 | 1.1325 | 0.459 |
| IP + MR | 1 | 1 | 0.14 | 0.4538 |
| RH + MR | 0.8188 | 0.9937 | -18.6175 | 0.317 |
| Greedy | 1 | – | -1.26 | 0.415 |



(a) CQL-R



(b) CQL*

Figure 3: Example trajectories generated by CQL-R and CQL* on an instance of **Unbalanced Cross**. Each color arrow represents the agent navigating to a new item. We can see that CQL-R first collects the carrot and cherry (which are low value) before finally making its way to the banana (which is the high value item in this episode). On the other hand, CQL* also collects the carrot first but then navigates to the banana afterward since it used the optimal policy for BC and has learned to better select high value items. As it's navigating to the banana, it's able to recall (represented by the gap in the two purple arrows) to shorten the distance needed to get there.

policy, which likely introduced instability to our learning, but we suspect that such an agent would perform poorly in certain environments where the item distributions have large variance.

One of the largest challenges we had during our time working on this project was that our gymnasium environment was a little too hacky–had it been better structured, we would have caught several bugs that we found late in the quarter. However, we still hope that our environment poses an interesting (and surprisingly difficult) problem that can be used in the future as a potential baseline for adaptive agents to solve. The sort of adaptation that we target is ubiquitous in everyday human life and would certainly be a great feat for an AI to overcome.

In the future, we would like to explore possibilities of expanding our environment to a continuous action space, which would reflect a more applicable, real-life task, such as that of a household robot. Further, we hope to further explore the idea of suboptimality detection in the environment: incorporating humans in the loop to give advice or simply developing a better recovery policy would pose an interesting challenge. Finally, we believe our recalling action was an interesting addition that made our training more difficult but allows for much more nuanced policy design.

## 7 Team Contributions

- **Denis:** wrote model architectures, created initial environment, ran experiments, large refactors
- **Victor:** environment work involving item distributions/randomizations necessary for experimental work, wrote evaluation code, manual agents (optimal/greedy), ran experiments, some model tweaking

**Changes from Proposal** As mentioned in the project milestone, we generally changed our proposal work, though the original inspirations remained the same. The reason for this is that our proposal was not well-defined and vague, and we only formalized our problem setup afterwards.

## References


Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5048–5058. https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. 2018. Exploration by Random Network Distillation. *CoRR* abs/1810.12894 (2018). arXiv:1810.12894 http://arxiv.org/abs/1810.12894

Maximilian Du, Alexander Khazatsky, Tobias Gerstenberg, and Chelsea Finn. 2024. To Err is Robotic: Rapid Value-Based Trial-and-Error during Deployment. arXiv:2406.15917 [cs.RO] https://arxiv.org/abs/2406.15917

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. 2016. RL$ˆ2$: Fast Reinforcement Learning via Slow Reinforcement Learning. *CoRR* abs/1611.02779 (2016). arXiv:1611.02779 http://arxiv.org/abs/1611.02779

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1126–1135. http://proceedings.mlr.press/v70/finn17a.html

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html

Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. 2020. Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices. *arXiv preprint arXiv:2008.02790* (2020).

Moinak Maiti. 2021. *Geometric Brownian Motion*. 67–88. https://doi.org/10.1007/978-981-16-4063-6_3

Robert C. Merton. 1976. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics* 3, 1-2 (1976), 125–144.

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

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