

Supporting Information for

Preemptive Solving of Future Problems: Multitask Preplay in Humans and Machines

Wilka Carvalho, Sam Hall-McMaster, Honglak Lee, Samuel J. Gershman

Wilka Carvalho.
E-mail: wcarvalho@g.harvard.edu

This PDF file includes:

Supporting text
Figs. S1 to S32
Tables S1 to S2
SI References

Supporting Information Text

Implementation details

We build our model closely following the Recurrent Replay Q-network architecture (1) from the ACME codebase (2). We begin with default hyperparameters from ACME and change them to achieve optimal training task performance. Optimal training performance is readily attainable across all tasks we study. We base our Jax implementation on PureJaxRL (3). We summarize the network architecture used by Q-learning, Dyna, and Multitask Preplay in Figure S28. We present shared hyperparameters across all models in Table S1 and model-specific parameters in Table S2.

Q-Networks. *JaxMaze.* Each cell of the grid carries one of $D = 6$ categorical labels (wall, empty, and the four object types that serve as goals). We embed each label into a 64-dimensional vector using the standard word-embedding architecture from NLP (4, 5); here the "words" are cell types, so the embedding lets the network share representations across grid positions that contain the same object. We use only the *initialization* scheme from that literature—the embedding matrix is trained from scratch via the RL loss, not loaded from any pretrained model. The per-cell embeddings are flattened into an $H \times W \times 64$-dimensional vector. The goal is a one-hot vector over the $C = 4$ goal-object types, processed through a 128-unit linear layer. We concatenate the observation and goal embeddings and give them as input to a 256-unit Long short-term memory (LSTM) (6). We leverage an LSTM, because it facilitates passing gradients through long time-series data such as our replay data. This dramatically improves learning of recurrent Q-networks. The output of the LSTM is concatenated with the goal embedding and passed through a 1024-unit MLP with 2 hidden layers and outputs A action values. All layers use ReLU activations and omit bias terms. *Craftax.* Craftax already provides a 56-dimensional continuous observation (normalized to $[0, 1]$) that summarizes the agent's local view and inventory, so no embedding step is needed—we pass the observation directly into the LSTM alongside the goal encoding. The goal dimensionality depends on the variant: $C = 3$ for the goal-conditioned Craftax used in the human experiments (three goal objects) and $C = 72$ for the standard Craftax achievement space used in the AI experiments (one dimension per achievement). We do not omit bias terms in this architecture.

Successor Feature Networks. These networks share the Q-network architecture but output $A \times C$ successor feature values instead of A Q-values. Action values are computed via per-action dot products with the goal vector.

Training. We trained all models with a protocol designed to mirror the structure of our human experiments, so that no model enters the test phase with an unfair informational disadvantage. For each JaxMaze experiment, we train on rotations of the same mazes used with human subjects (Figure S9 and Figure S10), and we swap which objects serve as train and test objects across runs so that every object serves as a train object at least once. We additionally train jointly on the two-paths experiment, the shortcut experiment, and a practice environment in which the agent must simply obtain all four objects within 10 timesteps. Thus every model has direct experience attempting to obtain the test-task objects, and this is reflected in test-task success being well above zero for USFA and HER (Figure S4 and Figure S15) despite the difficulty of our long-horizon tasks. For Craftax, each block uses a distinct procedurally generated map (Figure S19), and the same train/test-object swapping applies.

In JaxMaze, all models use $N = 32$ parallel environments. In Craftax, model-free methods (Q-learning, USFA, HER) use $N = 128$ parallel environments, while model-based methods (Dyna, Multitask Preplay) use $N = 32$ parallel environments. Models begin performing gradient updates after 10,000 data points are in the replay buffer. Model-free methods update every 160 environment steps across both domains. Dyna and Multitask Preplay update every 160 environment steps in JaxMaze and every 32 steps in Craftax; we tried updating model-free methods more frequently in Craftax but found this led to degraded performance. All models select actions with ϵ-greedy exploration, where each parallel environment is assigned its own ϵ that is held constant throughout training. In JaxMaze, ϵ values take the form 0.1^x with x linearly spaced on $[1, 3]$, placing ϵ in $[10^{-3}, 10^{-1}]$ with a bias toward the low end. In Craftax, longer episodes required wider exploration, so ϵ values take the form 0.1^x with x linearly spaced on $[0.05, 0.9]$, placing ϵ in $[\approx 0.13, \approx 0.89]$.

Experience Replay. We employ prioritized experience replay (7), which samples training transitions based on their TD error magnitude rather than uniformly. This allows agents to learn high-error experiences more quickly. The importance sampling exponent (we use 0.6) moderately corrects for the bias introduced by non-uniform sampling, while the maximum priority weight (we use 0.9) prevents any single high-priority transition from dominating gradient updates. We use batch sizes with $N = 32$ trajectories of length $T = 40$ for both JaxMaze and Craftax. One difference is that we find longer trajectories of length $T = 80$ improve performance of model-free methods in Craftax.

Counterfactual simulation. Both Dyna and Multitask Preplay use oracle world models. The key difference between real data and simulated data is that simulated data must begin from some starting "replayed" state and can only do a finite number of simulation steps. We set this to 15 for JaxMaze and 20 for Craftax. We have simulation policies use epsilon values distributed logarithmically between 1 and 3 using a base of 0.1, which effectively keeps values in the range of $[0, .1]$ with a bias for smaller values.

Optimization. Temporal Difference learning methods use their own predictions when constructing prediction targets. This means that as a model learns, its prediction target is changing, which can make learning unstable (8). To mitigate this, we employ "target networks", where we have a second set of parameters that are used to compute prediction targets. These parameters are updated to the latest parameters every 1000 learning updates. We use the Adam optimizer (9) with default values except changing the epsilon value to 10^{-5}. This is the default in ACME and increases numerical stability by preventing division by very small numbers in the denominator of the parameter updates. We use learning rates of 0.001 for model-free methods and 0.0003 for model-based methods in JaxMaze; all methods use a learning rate of 0.0003 in Craftax. To prevent

updates with gradients that are too large, we clip gradients to have a maximum global norm (across all parameters) of 80.

Hyperparameter search

All searches were random searches. For each sweep we drew 50 trials, and each trial sampled one value independently from every axis's range. Training-task performance selected the winner. Sweeps followed an inheritance chain across algorithms: each sweep began from the nearest parent configuration, and any axis not re-searched kept its parent's value.

JaxMaze. We based Q-learning on the Recurrent Replay Q-network implementation from ACME (1, 2), replacing the default return estimator with $Q(\lambda)$ (10) to better support the long horizons of our mazes. We searched the target-network update interval over $\{100, 200, 1000, 2000\}$; the learning rate over $\{10^{-2}, 3 \times 10^{-3}, 10^{-3}, 3 \times 10^{-4}\}$; the Adam numerical-stability constant over $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$; the Q-network hidden dimension over $\{256, 512, 1024\}$; the number of Q-network hidden layers over $\{1, 2, 3\}$; the discount factor over $\{0.99, 0.991, 0.992, 0.993\}$; and λ over $\{0.3, 0.6, 0.7, 0.8, 0.9\}$.

USFA was initialized from Q-learning and inherited every choice except those specific to the successor-feature head. We re-searched the learning rate, the successor-feature hidden dimension, the number of successor-feature hidden layers, and the discount factor over the same grids.

HER was also initialized from Q-learning, and most axes transferred. The critical addition was conservative all-goals learning, which our ablation shows is necessary to generalize on the larger mazes (Figure S6). Following standard practice we re-searched the learning rate, and we additionally searched the conservative Q-learning penalty coefficient over $\{0, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.

Dyna was initialized from Q-learning and uses the same per-environment ϵ values for simulation as for online acting. We re-searched the learning rate, Q-network hidden dimension, number of Q-network layers, and discount factor over the same grids.

Multitask Preplay was initialized from Dyna. We searched the conservative Q-learning penalty coefficient and the conservative all-goals coefficient, each over $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.

Goal-conditioned Craftax. Each algorithm began from its JaxMaze counterpart, with two changes that cut across every sweep. First, we re-searched the scaling axes that matter more in Craftax: Q-network hidden dimension over $\{256, 512, 1024\}$, number of hidden layers over $\{1, 2, 3\}$, number of parallel environments over $\{32, 64, 128\}$, trajectory length over $\{40, 80\}$, and batch size over $\{32, 64, 128\}$. Increasing hidden dimension, depth, parallel environments, and trajectory length each contributed substantially. Second, we replaced the JaxMaze per-environment ϵ scheme with the wider Craftax scheme described in the Implementation details.

USFA inherited the Q-learning improvements and we re-searched only the learning rate. HER inherited from its JaxMaze counterpart together with the Craftax Q-learning improvements; no further search was needed. Dyna inherited from Craftax Q-learning and additionally searched the simulation length over $\{10, 15, 20\}$, the Q-network hidden dimension, and the number of hidden layers. Multitask Preplay began from Dyna and searched the conservative Q-learning penalty coefficient over $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.

Standard Craftax. Standard Craftax shares its scaling regime with goal-conditioned Craftax, so most configurations transferred directly. Q-learning with the 1-step auxiliary task and Q-learning with the successor-feature auxiliary task each began from goal-conditioned Craftax Q-learning and searched the auxiliary-task coefficient over $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. HER, Dyna, and Multitask Preplay inherited their configurations from the goal-conditioned setting.

Environments

JaxMaze. *Tasks* involve controlling a red triangle to obtain objects in the maze (see Figure 3). The red triangle can move in any cardinal direction that is not blocked by a wall. When the red triangle moves over an object, it immediately collects it and the episode terminates. *Observations* of the environment contain the full map. Humans observe pixels and all models get symbolic representations of the entire map. *Actions*: Agents can move up, down, left, and right. *Rewards*: When a task's goal object is collected, agents get a reward of 1; all other timesteps get a reward of 0.

Goal-conditioned Craftax. This domain is a variant of the original Craftax environment (11). *Tasks* involve obtaining diamonds, sapphires, or rubies. Once a stone is collected, the episode terminates. *Observations* of the environment contain either the full map or a zoomed-in view (see Figure 4). During training, human subjects observe the full map and the zoomed-in view; during testing, they only see the zoomed-in view. All models only see the zoomed-in view during both training and testing. Models get symbolic representations of the entire map. *Actions*: Agents can move up, down, left, right, or select a "do" action which will collect the object in front of an agent. *Rewards*: When a task's goal object is collected, agents get a reward of 1; all other timesteps get a reward of 0. To make the environment easier for agents to learn in 1 hour, we start agents with a large "strength"—i.e. with the tools necessary to mine their goal object (e.g., a pickaxe), and made parts of the map that would otherwise need to be destroyed (such as non-precious stones) passable.

Experiment design

Sample sizes were determined based on pilot experiments (see Reporting summary). We implemented experiments using NiceWebRL*.

All experiments used the following block design. An experiment consisted of 4 blocks, where each block had an associated maze (see examples of environments and goal/start sampling in Figure S9 and Figure S10). For JaxMaze, each block within an experiment presented a rotation of the same maze. In Craftax, different blocks had different procedurally generated maps. Blocks were divided into a training segment where subjects learned tasks from $\mathbb{M}_{\texttt{train}}$ and a *subsequent* evaluation segment where subjects were tested on tasks from $\mathbb{M}_{\texttt{test}}$. During training, subjects had 80 attempts to successfully complete 8 trials per train task. During evaluation, subjects had 1 attempt for each evaluation task. Subjects were given a bonus for completing evaluation tasks more quickly but there was no time-limit. When a trial was completed, subjects had to press a key to move on to the next trial. Across subjects, block ordering and object categories were randomized. The order that tasks were presented was also randomized.

JaxMaze: Two Paths experiment (experiment 1) and Shortcut experiment (experiment 2). Both experiments followed the same protocol but differed in the mazes presented to subjects. Individual blocks consisted of 2 training tasks and 1 evaluation task. During training, subjects were told what task they would later be evaluated on and their starting location was chosen from 8 possible locations. Subjects were told that they would receive a bonus for completing the evaluation task in under 20 seconds.

JaxMaze: Intermediary start experiment (experiment 3) details. Individual blocks consisted of 2 training tasks and 2 evaluation tasks. During training, subjects were told what task they would later be evaluated on. Subjects were told they would get a bonus for completing evaluation tasks in under 20 seconds.

JaxMaze: Juncture experiments (experiment 4). Individual blocks consisted of 1 training task and 2 evaluation tasks. During training, subject's starting location was chosen from 8 possible locations. Across all experiments, subjects were spawned in a novel starting location during evaluation. We tested 3 conditions in these experiments: (1) the test object is nearby a training path and known during training (`near`, `know`; Figure 3 E), (2) the test object is nearby a training path and not known during training (`near`, `unknown`; Figure 3 F), and one where the test object is "far" from a training path and known during training (`far`, `know`; Figure 3 G). For each condition, there were 2 evaluation tasks which had the same optimal path as a solution—but in different parts of the map. For the condition when an object was nearby but unknown, we ensured that subjects could not guess which object they would be evaluated on by having two potential goal objects on opposite sides from the spawning location. When the test object was nearby, subjects were told they would get a bonus for completing evaluation tasks in under 3 seconds; when it was further away, we changed this to 10 seconds.

Craftax experiment (experiment 5). Individual blocks had a unique map (Figure S19), where we placed 2 train objects in distinct locations and then a test object visibly nearby 1 of the train objects. Like in our JaxMaze experiments, we chose start and end positions such that there were two paths to the test object—where one overlapped with the optimal path to the train object and one that did not. To get maps with this property, we procedurally generated 100 worlds and manually chose from this set. To create this gap, we exploited that subjects cannot pass through lava or water and thus would have to go around, creating two paths to objects. Blocks were divided into a training segment where subjects learned 2 training tasks from $\mathbb{M}_{\texttt{train}}$ and a *subsequent* evaluation segment where subjects were tested on 1 evaluation task from $\mathbb{M}_{\texttt{test}}$. We ran two conditions: one where subjects were told what task they would later be evaluated on and one where they were not. In each training episode, their starting location was chosen from 4 possible locations. Subjects were told they would get a bonus for completing evaluation tasks in under 20 seconds.

Statistical Analyses

Path reuse (experiments 1, 2, and 5). To test whether people demonstrated above chance evidence of partial path reuse, we first computed the proportion of trials in which each subject reused the prior training path at evaluation. We note that the exact functions for computing path reuse for JaxMaze and Craftax were not preregistered, though the statistical tests we used were. *JaxMaze* We define path reuse as follows. We have two binary matrices $M_{\texttt{train}}$ and $M_{\texttt{test}}$ of the same dimension $H \times W$ that have $M(i,j) = 1$ if an agent was in position i, j. We compute path overlap o of how much a participant's test path overlaps with their training path by computing how much of the path is shared, divided by the length of the test path, i.e. $o_{\texttt{map}} = \frac{M_{\texttt{train}} \odot M_{\texttt{test}}}{||M_{\texttt{test}}||}$, where $\odot$ is an element-wise product and $||M||$ defines the norm of matrix M. Note that $o_{\texttt{map}} \in (-1, 1)$. Path reuse is then $p_{\texttt{reuse}} = o_{\texttt{map}} > \alpha_{\texttt{map}}$ where $\alpha_{\texttt{map}}$ is a threshold. We set $\alpha_{\texttt{map}} = .5$ for prediction 1 and $\alpha_{\texttt{map}} = .8$ for prediction 2.

For prediction 1 our conclusions are robust to the choice of $\alpha_{\texttt{map}}$ (Figure S11 and Figure S12), because there are essentially only two paths to the test object. For prediction 2, shortcuts open up many more possible routes, so the choice of threshold matters more. We settled on $\alpha_{\texttt{map}} = .8$: test paths that partially took a new shortcut fall below this value (Figure S13, top row), while overlap above .8 indicates that the subject took no shortcut at all, i.e. completely reused the prior path (Figure S13, bottom row).

Craftax is a more open-ended environment where many paths can be taken between two points. To account for this, we have $M_{\texttt{train}}(i,j) = 1$ for all points along a person's train path or points 1 position away (i.e. $i \pm 1$ or $j \pm 1$). It is possible for there to be a relatively high overlap but for the participant to come in from a different angle, so we additionally measure the cosine similarity for the last 10 positions of a person's path. Let p be a vector of positions a person has taken. Then we compute

*https://github.com/wcarvalho/nicewebrl/

direction overlap via cosine similarity as $o_{\texttt{dir}} = \frac{v_{\texttt{train}}^{\top} v_{\texttt{train}}}{||v_{\texttt{train}}|| \cdot ||v_{\texttt{test}}||}$ where $v = \frac{1}{10}\sum_{t=T-11}^{T-1}(p_{t+1} - p_t)$ and $x^{\top}y$ is a dot-product (we repeat this for both train and test). Note that $o_{\texttt{dir}} \in (-1, 1)$. We had path reuse be 1 if there was a sufficient overlap for both measures or if map path reuse was very high. That is, path reuse is then $p_{\texttt{reuse}} = (o_{\texttt{map}} > \alpha_{\texttt{map}} \wedge o_{\texttt{dir}} > \alpha_{\texttt{dir}}) \vee (o_{\texttt{map}} > 2 \cdot \alpha_{\texttt{map}})$ where $\alpha_{\texttt{map}}$ is a direction threshold. We used $\alpha_{\texttt{map}} = .25$ and $\alpha_{\texttt{dir}} = .5$.

To build intuition for this criterion, Figures S20–S22 show representative evaluation trials. Sometimes a subject follows the training path but diverges slightly, so map overlap stays modest (e.g. $o_{\texttt{map}} = .316$ in Figure S20a) even though the angle of approach to the object is preserved across train and test; the direction clause recovers these cases. In other cases the approach is nearly perpendicular across train and test, yet the paths overlap heavily (Figure S20b), and the high-overlap clause fires. Figure S21 contrasts a trial where both clauses are satisfied with a false positive. Figure S22 shows a true negative alongside a near-miss false negative, in which the subject broadly reuses the training path but falls short on both overlap and approach angle.

One way to check the geometric criterion is to replace it with a method that shares none of its assumptions. We re-classified every evaluation trial with a vision-based language model (Claude Sonnet 4.5), which inspected side-by-side renderings of the train and test paths (Figure S18). The model is more conservative — 63.5% reuse when the goal was known and 63.7% when it was unknown, against roughly 70% under the geometric criterion on the same cohort ($n = 359$ and $n = 344$ trials, 100 subjects per condition). Critically, the two methods' 95% confidence intervals overlap in both conditions (Figure S17), placing the two estimates within sampling error of one another.

We then tested whether the resulting distribution was normally distributed using a Shapiro-Wilk test. As the data were not normally distributed in all cases (all W-values > 2964 and all p-values $< 9.4 \times 10^{-9}$ for JaxMaze; all W-values > 2966 and all p-values $< 1.3 \times 10^{-10}$ for Craftax), we tested whether the proportion of evaluation trials with partial reuse was significantly greater than 0.5, using non-parametric, one-sided Wilcoxon signed-rank tests.

RTs during partial reuse (Figure 3 C&D). An episode of T steps contains T response time (RT) measurements. Each RT measurement corresponds to the time between when an image is seen by a subject and when they hit a key in response. RT measurements have a skewed distribution. A standard method to deal with this is to log-transform RTs so that they approximately satisfy Gaussian distributional assumptions (12, 13). Thus, we applied all statistics to log-transformed RT measurements. In this analysis, we either look at the maximum RT value across all T steps or the very first one. To evaluate log RTs on reuse and non-reuse choices, we used linear mixed effects models (LMEs) with the following equation: $RT = B0 + B1 \times p_{\texttt{reuse}} + B2 \times (1|\text{subject ID}) + \epsilon$. In this equation, B0 is a constant and ϵ is residual error. The fixed effect (path score) is a binary variable indicating whether subjects partially reused a training path (1) or followed a new path (0) on a given evaluation trial. The random effect (1|subject ID) models random intercepts for each subject. LMEs were used in these cases because the path score measure was choice dependent, as opposed to being manipulated experimentally. A significant B1 coefficient with a negative sign indicates significantly lower RTs when subjects partially reused the training path.

RTs during juncture and starting point manipulations (Figure 3 H&J). To test whether people were faster when placed along the training path at evaluation, we first computed the log RT for the training junction and non-training junction evaluation trials. We then took the difference in log RT between these evaluation trials for each subject and tested whether the resulting distribution was normally distributed using a Shapiro-Wilk test. As the data were not normally distributed, we used one-sided Wilcoxon signed-rank tests to test whether subjects were significantly faster when placed along the training path at evaluation. An equivalent statistical analysis was used to compare RTs for on-path and off-path evaluation trials.

Craftax Environment Transfer Details

This is a procedurally generated environment where different initialization seeds spawn different environments. In the normal setup, all initialization seeds are used for training. To enable the study of transfer across environments, we followed a standard protocol in the literature (14, 15) and designated $n_{\texttt{train}}$ train initialization seeds and $n_{\texttt{test}}$ test initialization seeds. For each evaluation we subsample a fixed set of seeds from the held-out test pool; the specific seeds are open-source and the subsampling is visible in the released analysis scripts on GitHub. Craftax is a **single-task** environment where there is a **single global reward function** that gives different priorities to different goals. Specifically, for m goals, $r_g = (\phi_{\texttt{achieved}}(s_t) - \phi_{\texttt{achieved}}(s_{t-1}))^{\top} w_g$ where $w_g \in \mathbb{R}^m$ is a vector that gives different preferences to different goals. We do not change this. In this article, our goal is to show that *preplay towards accessible goals in one environment improves generalization to a novel environment*. As such, we focus on the setting where agents know about what goals exist and when they are available. For example, if you see a diamond, this becomes an available goal for you when you already have the necessary tool (i.e., a pickaxe). We used an LLM to create a function which goes through the logic of Craftax and automatically computes this, showing that it is possible to automate these kinds of computations today with modern language models. We expose this to the agent with a function $\phi_{\texttt{avail}}(s) \in \mathbb{R}^m$, which is 1 for a goal if that goal can be completed. Index i of $\phi_{\texttt{avail}}(s)$ corresponds to the availability of the goal specified by index i of $\phi_{\texttt{achieved}}(s)$. We experimented with also exposing subgoals as accessible only if they were visible and not if preconditions were met. Multitask Preplay still outperformed Dyna in this setting (see Figure S29) but we chose to focus our results in the setting where—as is common with people—an agent is aware of possible subgoals and whether they are currently possible. This is because our focus is on how this *improves generalization to new environments*; not on how it improves learning in a training environment—something already shown in prior work (16).

To expose this knowledge to all agents, **observations** are defined as $x = (x_{\texttt{image}}, \phi_{\texttt{avail}}(s), \phi_{\texttt{achieved}}(s))$ where $x_{\texttt{image}}$ is a zoomed-in view of the agent in the map, $\phi_{\texttt{avail}}(s)$ describes which goals are achievable, and $\phi_{\texttt{achieved}}(s)$ describes which goals have been achieved thus far. All agents get all of this information. **Actions**. There are 74 actions which involve

object-interaction actions like `PLACE_PLANT` or `MAKE_WOOD_PICKAXE` which work successfully once relevant preconditions are met.

Model implementations We consider 4 models that each take advantage of $\phi_{\texttt{avail}}(s)$ in different ways. All models, including Dyna, use $\phi_g(s) = [\phi_{\texttt{avail}}(s), \phi_{\texttt{ach}}(s)]$ as input for their action-value predictions.

Q-learning + 1-step model. Here, we leverage $\phi_g(s)$ to define an auxiliary task where the agent learns to predict $\phi_g(s_{t+1})$. We use the same neural network as the Q-network architecture from the Craftax human experiments with the following change. We also produce $\hat{\phi}_g(s')$ with a 512-unit MLP that takes the LSTM state as input.

Q-learning + SF. Since this is a single-task environment, successor features can only be used as an auxiliary task and not as a method for transfer across tasks. Here, $\phi_g(\cdot)$ defines the features for successor features. Like with the previous baseline, we change the Craftax human experiments Q-network to additionally produce $\hat{\psi}^{\pi}(s, a, g)$ with a projection from the LSTM state as input.

Dyna. Dyna uses the learned world model to generate simulations that optimize the main reward R_g. Unlike Multitask Preplay, Dyna does not sample subgoals—all simulated experience is used to improve the policy for the single task objective. We also ablate a variant where the simulation goal is randomly replaced with a uniformly sampled goal from $\phi_{\texttt{avail}}(s)$ at varying rates. This tests whether naively introducing off-task simulations (without the structured all-goals learning of Multitask Preplay) can improve generalization (see Figure S30).

Multitask Preplay. Since this is a single task setting, we define the main goal as g and then sample subgoals g' during preplay. This algorithm works equivalently to in our previous experiments. The main difference is that we sample $g' \sim \text{Uniform}(\phi_{\texttt{avail}}(s))$. Note that a sampled g' will often correspond to a goal that was *not* completed in the real-life experience. We learn a single action-value function for both g and g' and simply change which is input depending on the relevant computation. This effectively allows Multitask Preplay to preplay simulations towards g' and then update action-values for both r_g and r'_g. We find that this is key to maximizing the benefits of multitask preplay (see Figure S29).

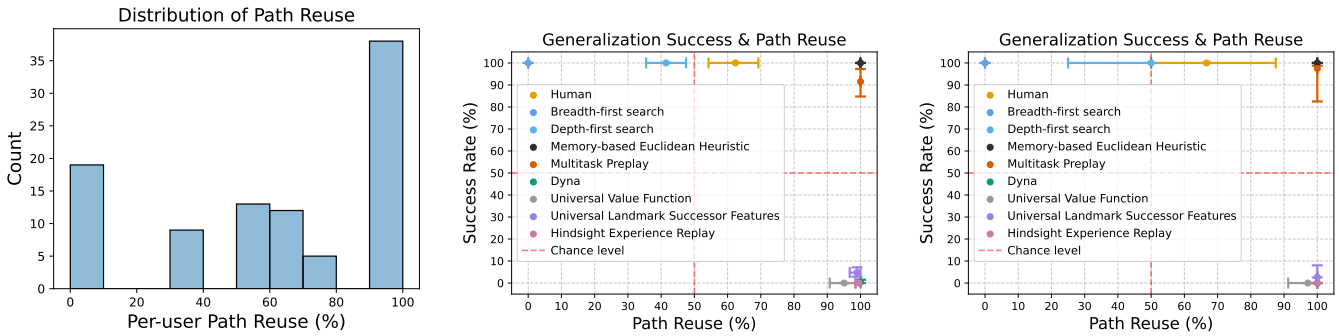


Fig. S1. JaxMaze Path Reuse supplementary results. (Left) Human rate distributions. (Center) Mean success rates. (Right) Median success rates.

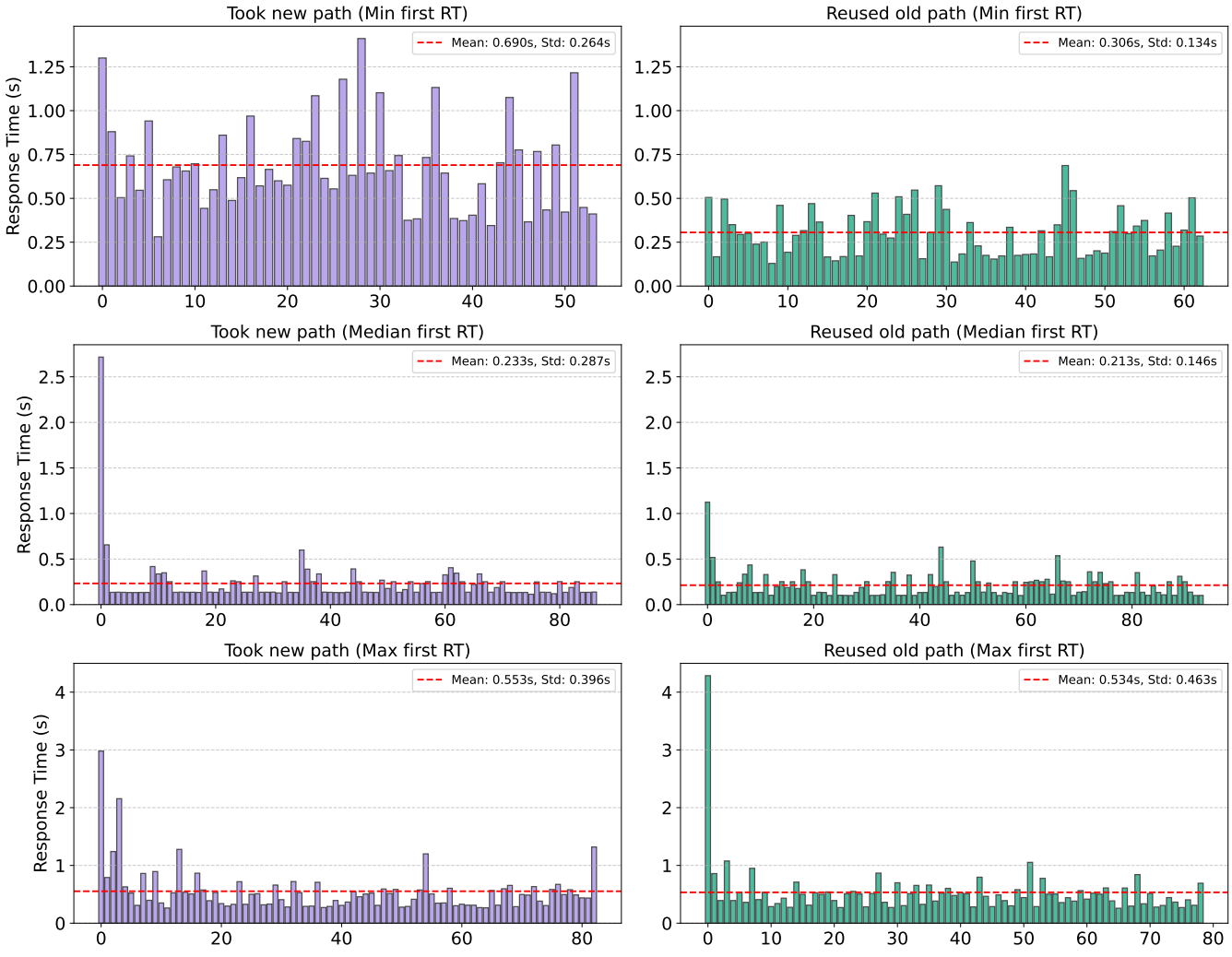
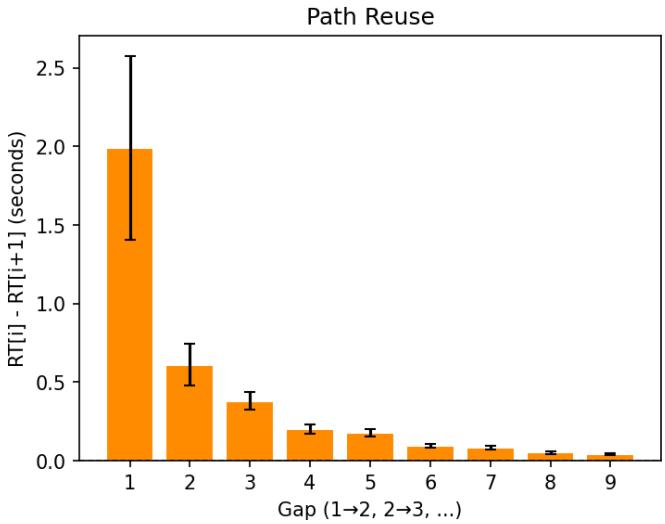
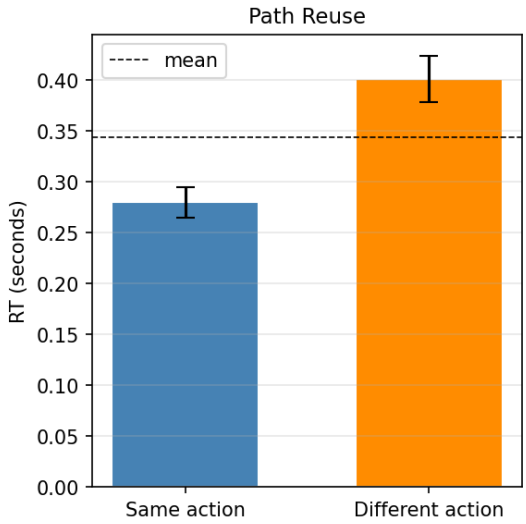


Fig. S2. Example differences in RTs when people take a new path (left) or reuse an old path (right).



(a) The relative difference between RT[i] - RT[i+1] for the top-10 RTs, averaged over both participants and trials. For every individual trial (i.e. episode) a participant completes, we compute this metric. We then average over both all trials and all participants. For each index, we present mean and standard error.



(b) The average response time when people take the same action as in the previous timestep or a different action. We sort RTs into whether the action taken is the same as the previous or different and present the mean and standard error over these sets.

Fig. S3. Characterizing the response time distributions. In (a) we show that, on average for every participant, the highest response times are **greater** than 2 seconds longer than the next highest response times. In (b) we show that participants take longer to select an action when the action is different from the previous timestep. Together, these results suggest that the bulk of "computation" (i.e. cognitive work) is concentrated in one or a few timesteps, but that there are other timesteps where participants consistently think longer (e.g. when they need to select a new action).



Fig. S4. JaxMaze training performance. Train and test success rates for all models on the JaxMaze environment.

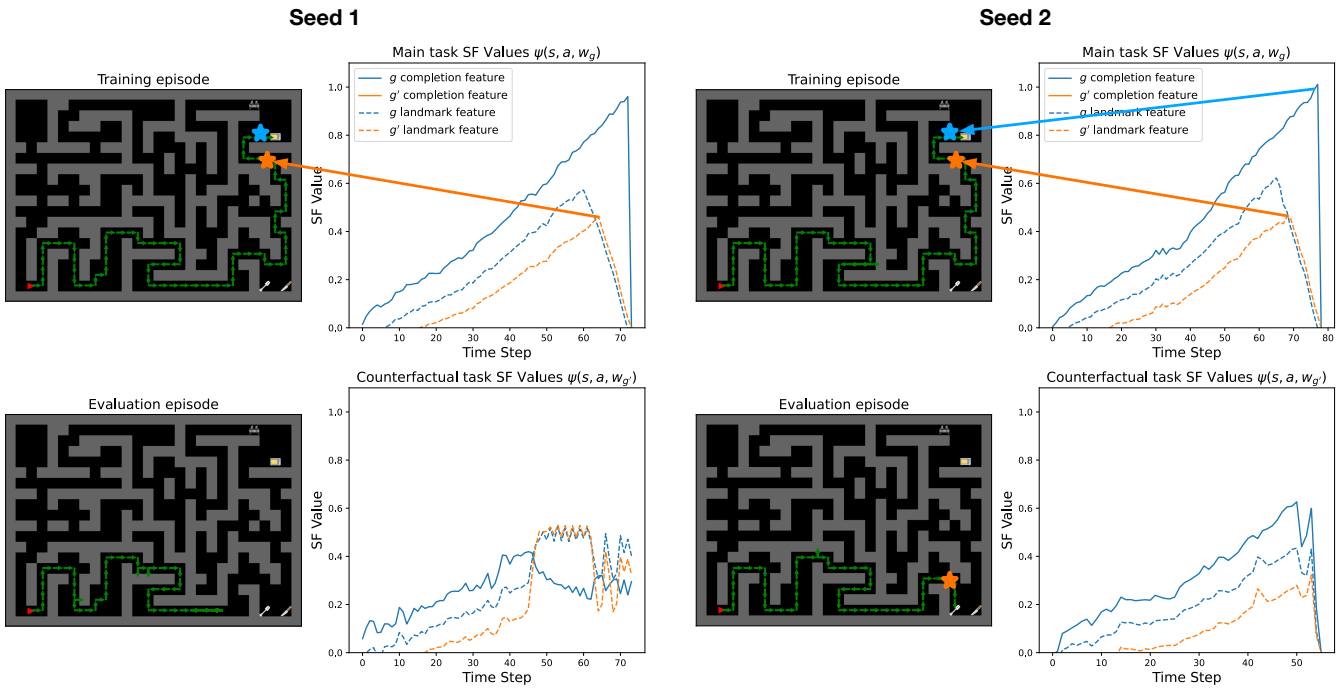


Fig. S5. Landmark successor features fail to generalize to "off-task" goals g' nearby main task goal g (Figure 3 A&B). Shown are two typical failure modes when $\psi(s, a, \cdot)$ predicts goal completion features and landmark features. **Top row**. The first mechanism for generalization is to select actions with high $\hat{\psi}^{\pi}_{\theta}(s, a, g)^{\top} w_{g'}$. We can see that, counterintuitively, landmark features lead to a corresponding "successor feature" (orange) with a maximum value at a position along a previous path that is *not closest* to g' but instead a position from which you will most often be "nearby" g'. To complete g', some form of planning is required. **Bottom row**. The other mechanism for generalization is to directly learn $\psi(s, a, g')$ with trajectories collected while pursuing g via off-task learning (17). While this is possible in principle, we identify two common failure modes: either the resulting estimate is noisy and uninformative (left) or $\hat{\psi}^{\pi}_{\theta}(s, a, g')$ ends up converging to the $\psi(s, a, g)$ for a high signal training goal g (right).

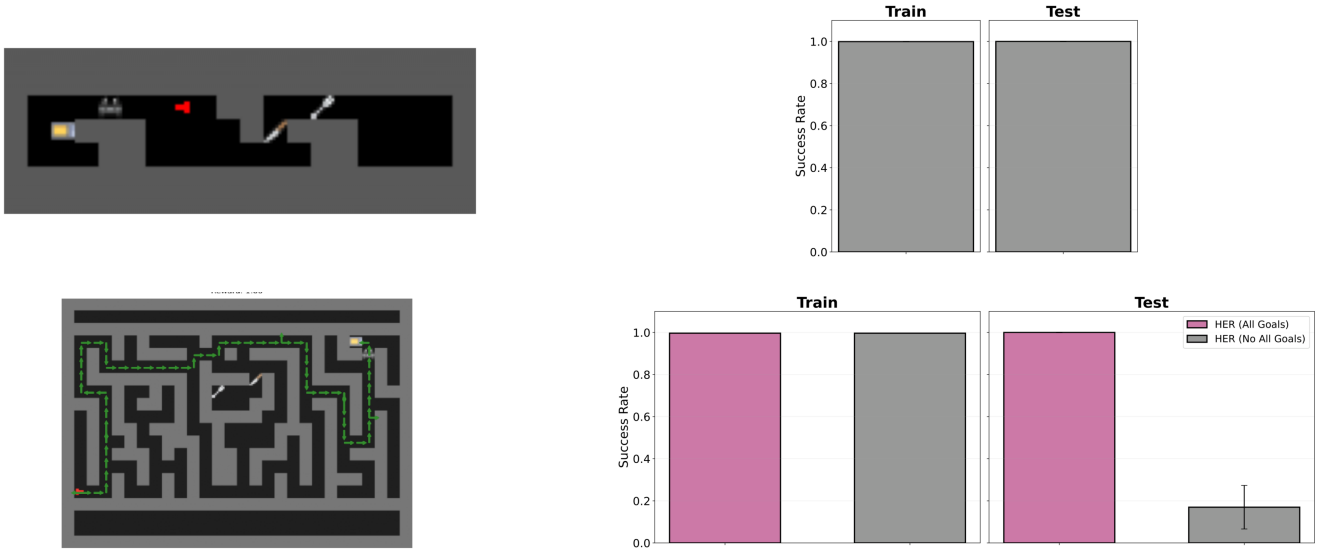


Fig. S6. HER with and without all-goals learning. Train and test success rates for Hindsight Experience Replay (HER) on the small (top) and large (bottom) JaxMaze environments. In the large environment, HER with all-goals learning substantially outperforms HER without all-goals learning on held-out test mazes, while both achieve near-perfect training performance.

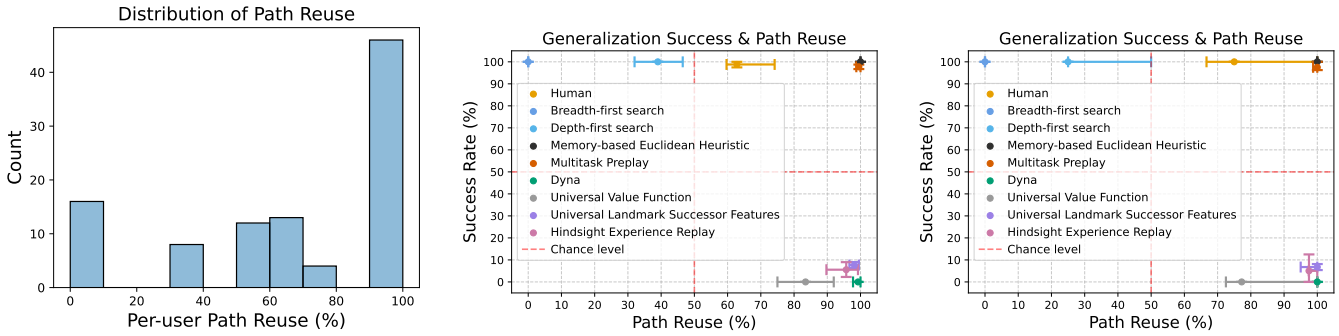


Fig. S7. JaxMaze Shortcut supplementary results. (Left) Human rate distributions. (Center) Mean success rates. (Right) Median success rates.



Fig. S8. Differences in RTs when people take a new path in new part of maze (left) or took a new path in a part of the maze where they could have used preplay (right). Example RTs over an episode during eval task 2 (left) or eval task 1 (right) in a setting where the goal object is unknown (Figure 3 E&H).

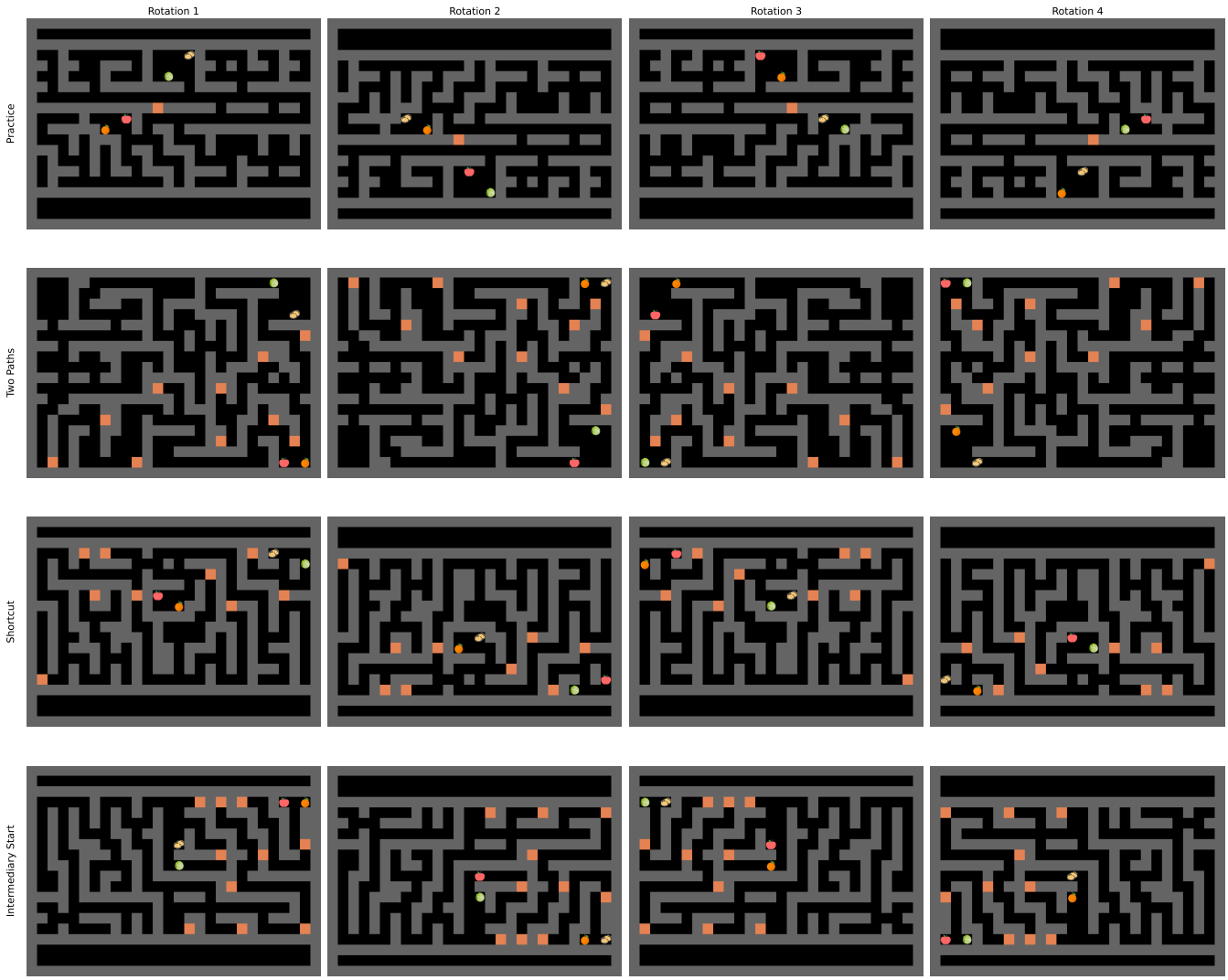


Fig. S9. JaxMaze environments used in experiments 1–3. Each column shows a different rotation of the same maze. Rows correspond to (top to bottom): the practice maze, the Two Paths experiment, the Shortcut experiment, and the Intermediary Start experiment. Object icons indicate goal locations and red squares indicate possible starting locations.

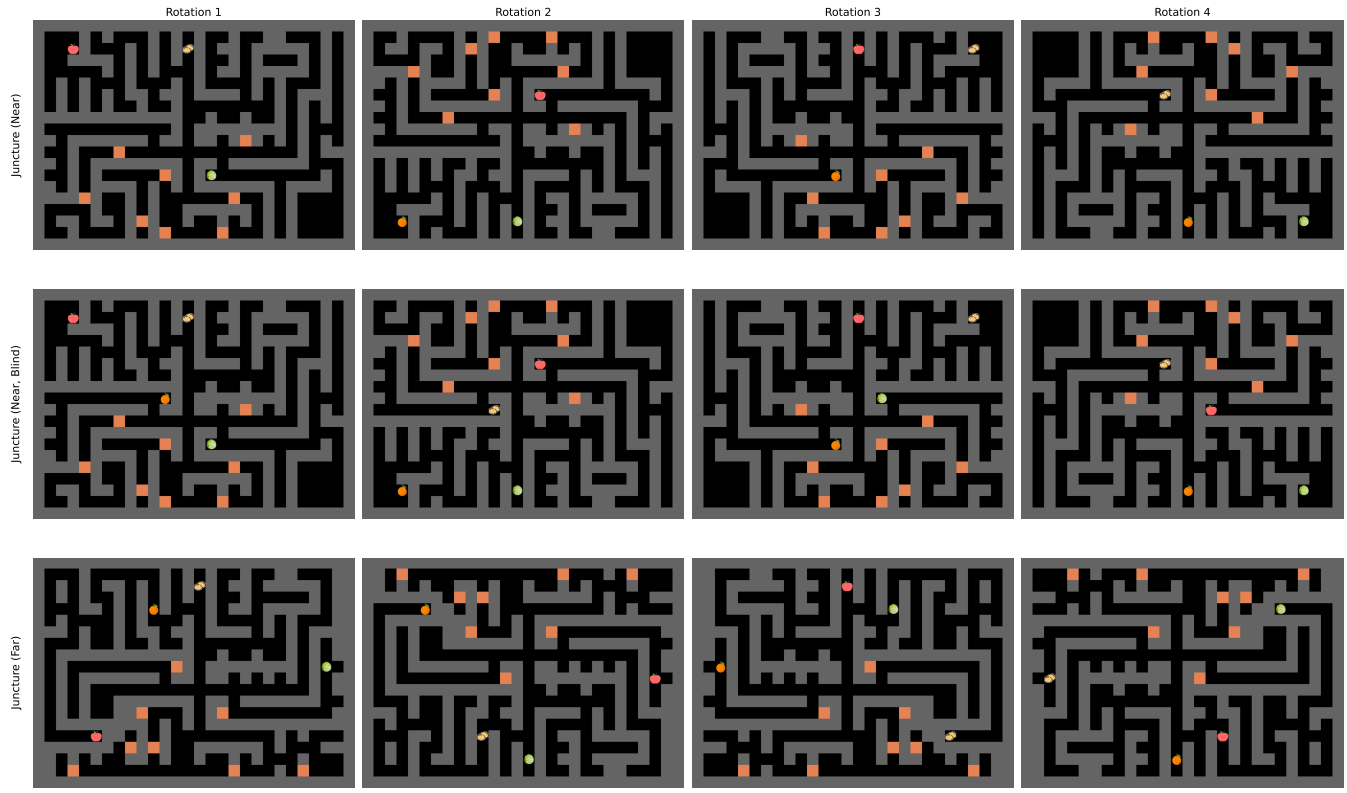


Fig. S10. JaxMaze environments used in the Juncture experiment (experiment 4). Each column shows a different rotation. Rows correspond to the three conditions: Near with known test goal, Near with unknown test goal, and Far with known test goal. Object icons indicate goal locations and red squares indicate possible starting locations.

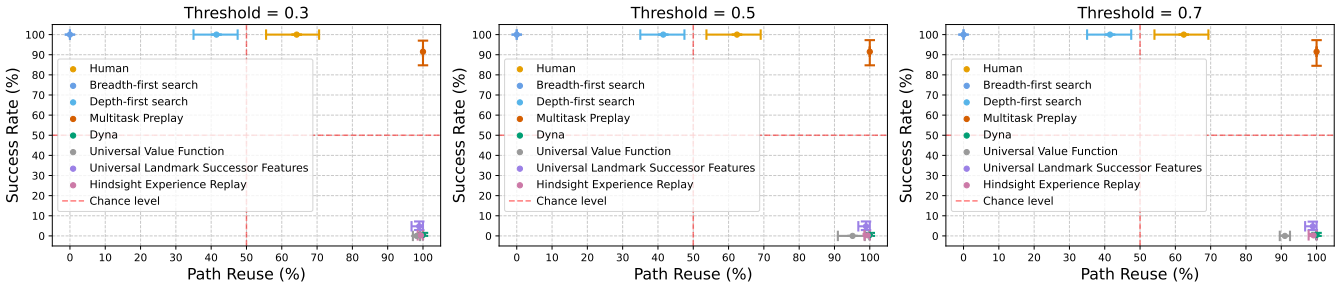


Fig. S11. JaxMaze prediction-1 path-reuse proportions are robust to the threshold α_{map}. Proportion of evaluation trials classified as path reuse across different α_{map} values. Because the JaxMaze environments used for prediction 1 admit essentially only two viable paths to each test object, the qualitative conclusions do not depend on the exact threshold choice.

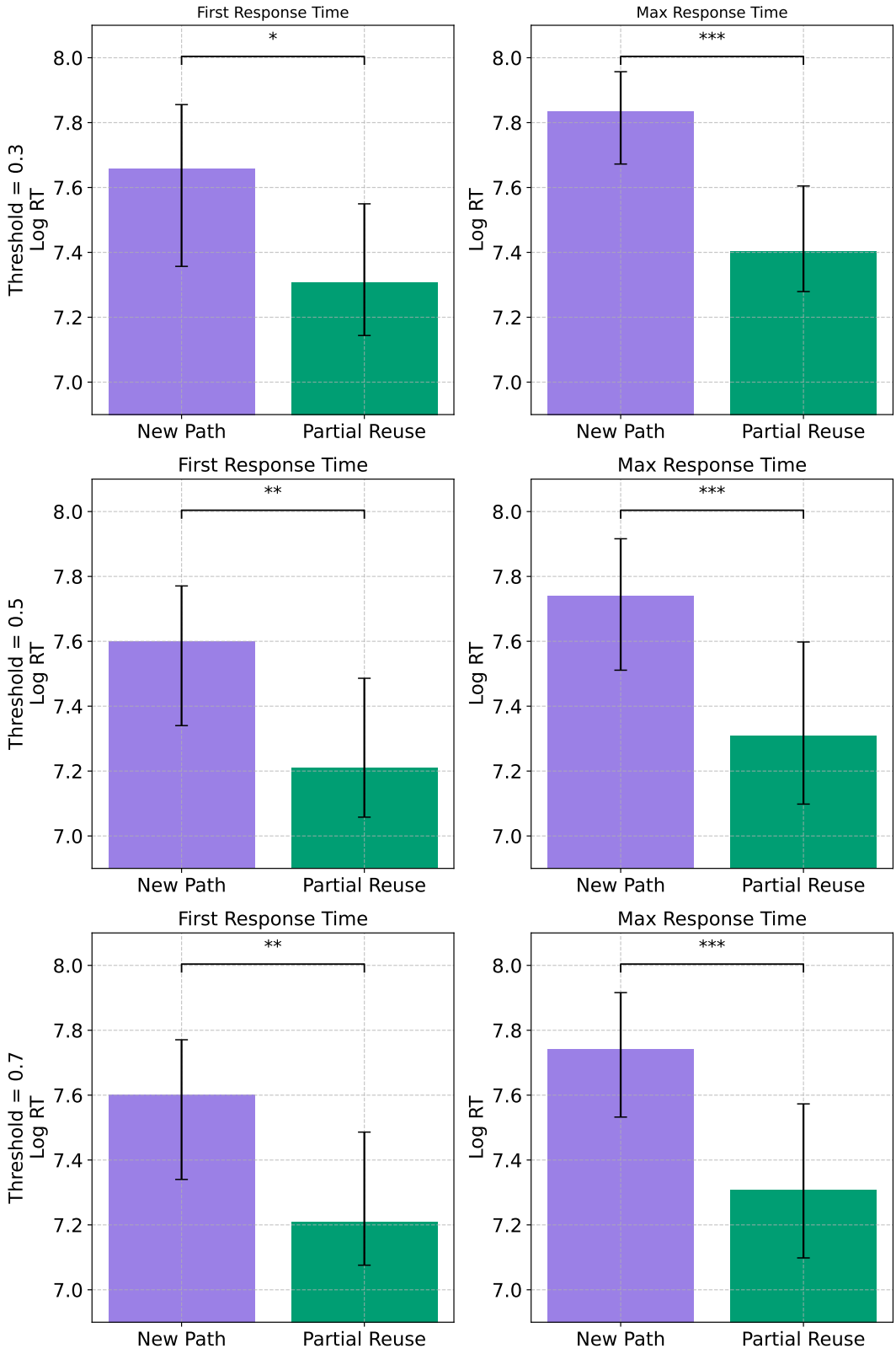


Fig. S12. JaxMaze prediction-1 RT effects are robust to the threshold α_{map}. Log-RT comparisons between path-reuse and new-path trials across different α_{map} values. The RT advantage for reused paths is preserved across thresholds.

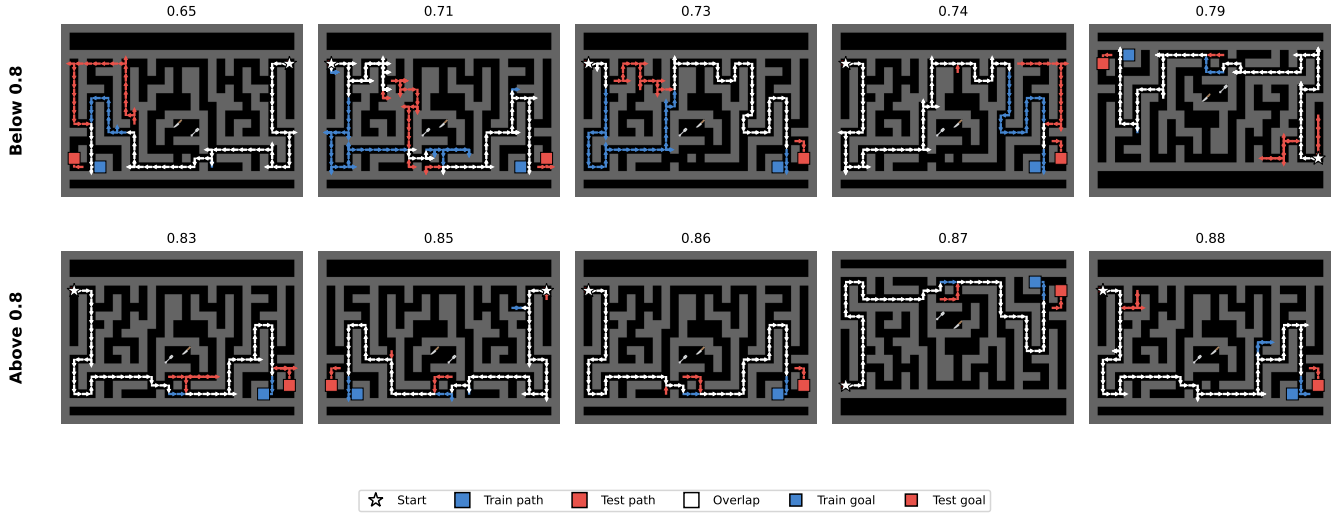
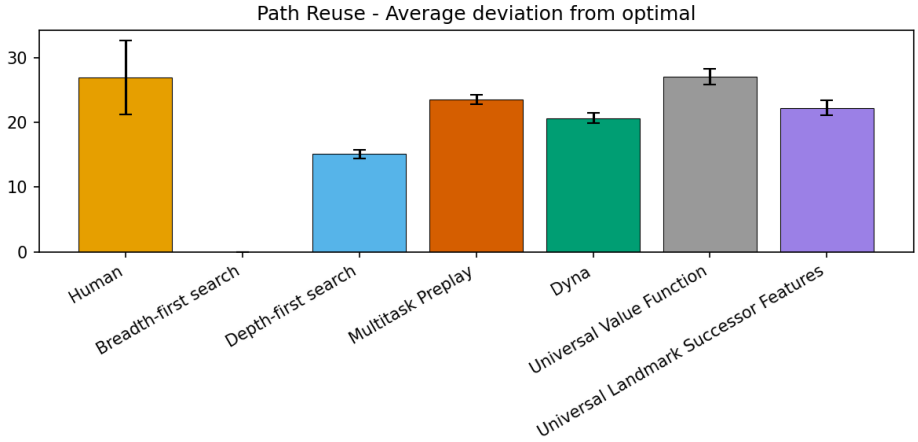
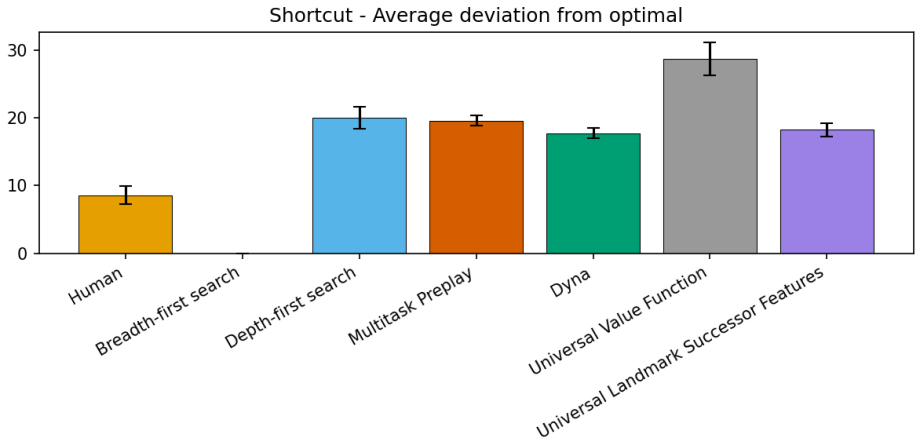


Fig. S13. The $\alpha_{\text{map}} = .8$ threshold separates shortcut-taking from non-shortcut behavior in JaxMaze prediction 2. Top row: evaluation paths that partially take a newly-opened shortcut produce overlap values that fall below $.8$. Bottom row: overlap values above $.8$ correspond to trials where the subject took no shortcut and reused the training path.



(a) Experiment 1 (Path Reuse)



(b) Experiment 2 (Shortcut)

Fig. S14. Optimal length deviations across experiments 1 and 2. Distribution of deviations from optimal path length for (A) the path reuse experiment and (B) the shortcut experiment.

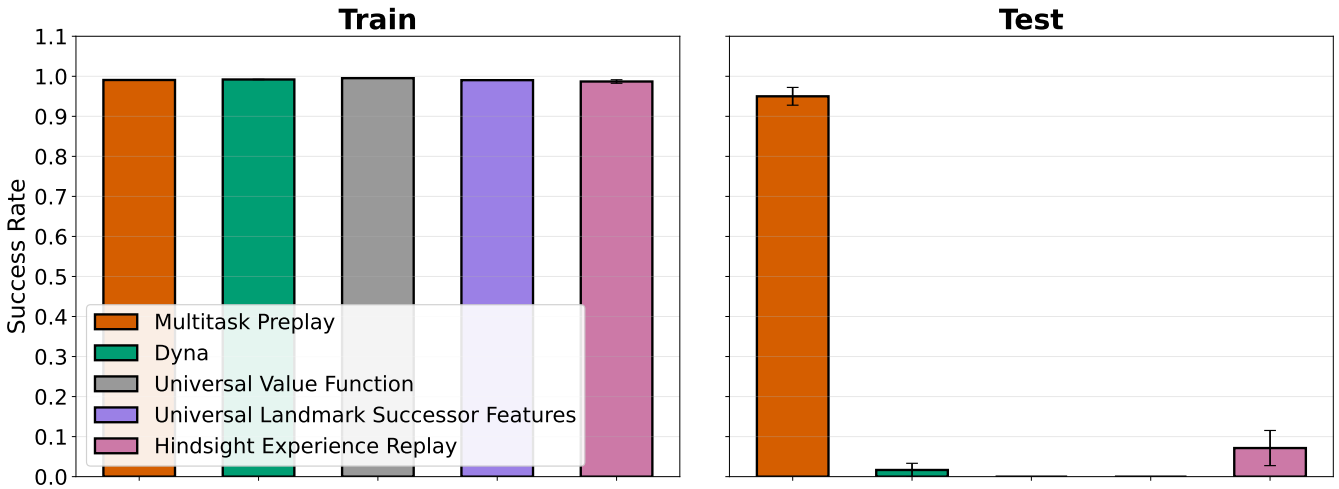


Fig. S15. Craftax training performance. Train and test success rates for all models on the Craftax environment.

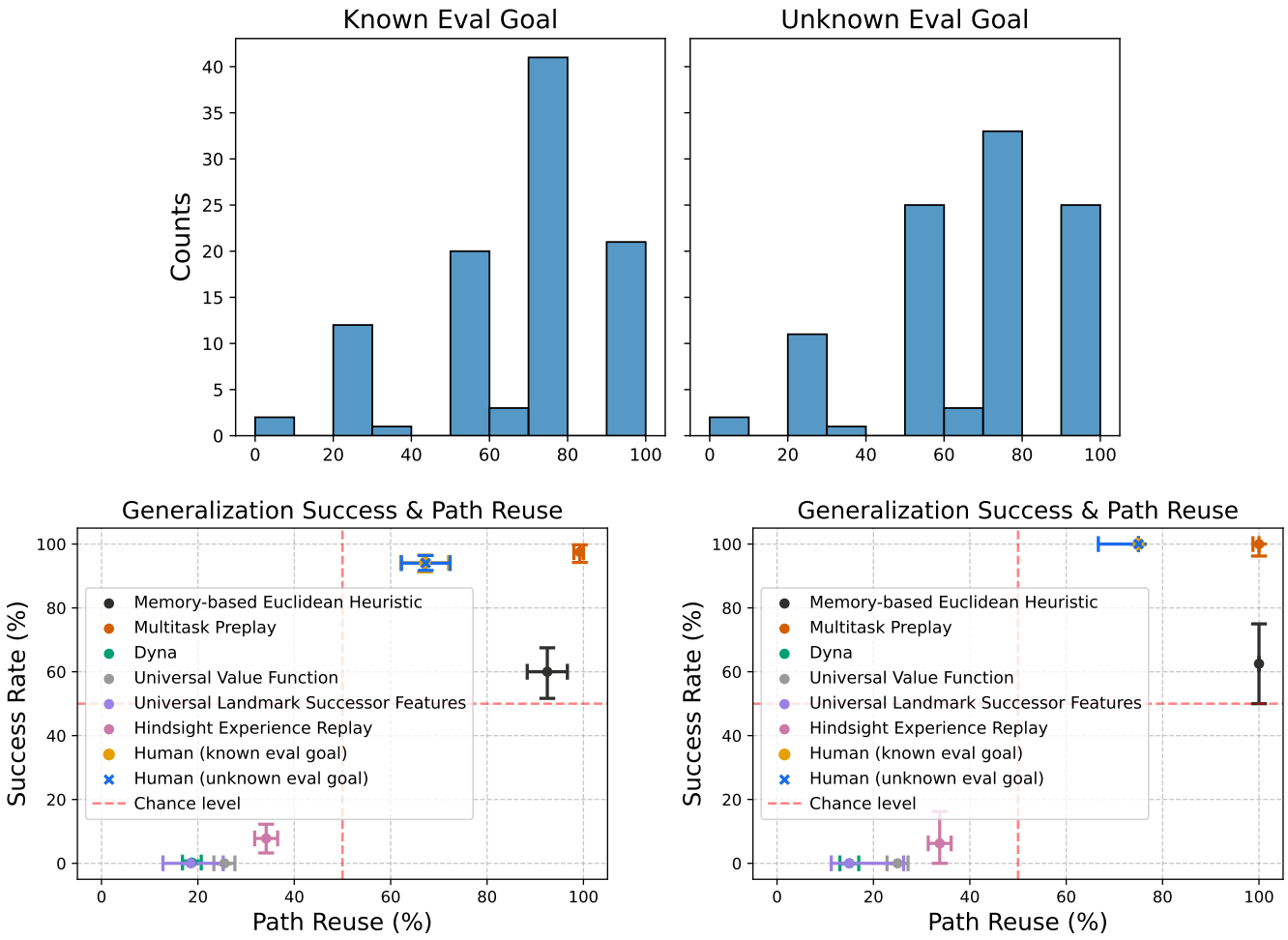


Fig. S16. Craftax Path Reuse supplementary results. (Top) Human path reuse rate distributions. (Bottom left) Mean success rates. (Bottom right) Median success rates.

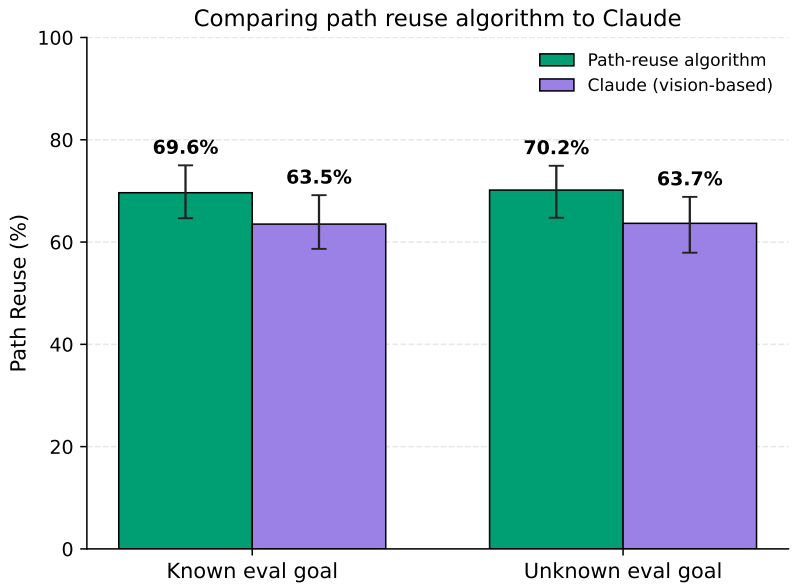


Fig. S17. Validating the Craftax path-reuse criterion with a vision-based language model. Path-reuse rates from the geometric criterion (green) and from a vision-based LLM (Claude Sonnet 4.5; purple) agree within error: the 95% confidence intervals overlap in both conditions, placing the two estimates within sampling error of one another. The LLM's classifications are uniformly more conservative (about 6–7 percentage points lower in point estimate) but remain well above the 50% chance baseline. Bars reflect the canonical cohort (first 100 subjects per condition; $n = 359$ trials with known goal, $n = 344$ with unknown goal).

Path-reuse classification prompt (Claude Sonnet 4.5)

```
You will see two images of the same Craftax world. In each, a red line shows
the path a participant took from the yellow star (start) to the yellow circle
(goal). Both paths are from the same person on different trials -- first
reaching a training goal, then reaching a test goal placed nearby. They
started in the same position both times.

Read both images first:
1. Read {train_path}
2. Read {test_path}

Did the participant reuse the training path on the test trial? Answer two
questions:

1. SAME SIDE OF WATER: For each meaningful body of water (blue), did both
   paths go on the same side of it?

2. SAME ROUTE: Do both paths follow roughly the same route to the goal and
   approach from roughly the same direction? (A brief detour where the test
   path wanders off and then rejoins the training route is fine.)

Output exactly ONE JSON object as the LAST line of your response, no code
fences. Label "path_reuse" only if both answers are yes; otherwise
"no_path_reuse". Include a brief reasoning (1-2 sentences) describing what
you saw on each side of each path:
{"same_side": "yes|no", "same_approach": "yes|no",
 "reasoning": "<1-2 sentences>",
 "label": "path_reuse|no_path_reuse"}
```

Fig. S18. Prompt used to classify Craftax path reuse with Claude Sonnet 4.5. Each evaluation trial was classified by sending Claude two rendered images of the participant's training and test paths and the prompt above. The placeholders `{train_path}` and `{test_path}` are replaced at inference time with file paths to the rendered images, which Claude reads via its vision-enabled file-reading tool. A trial is labeled `path_reuse` only if Claude answers "yes" to both the same-side-of-water and same-route questions.

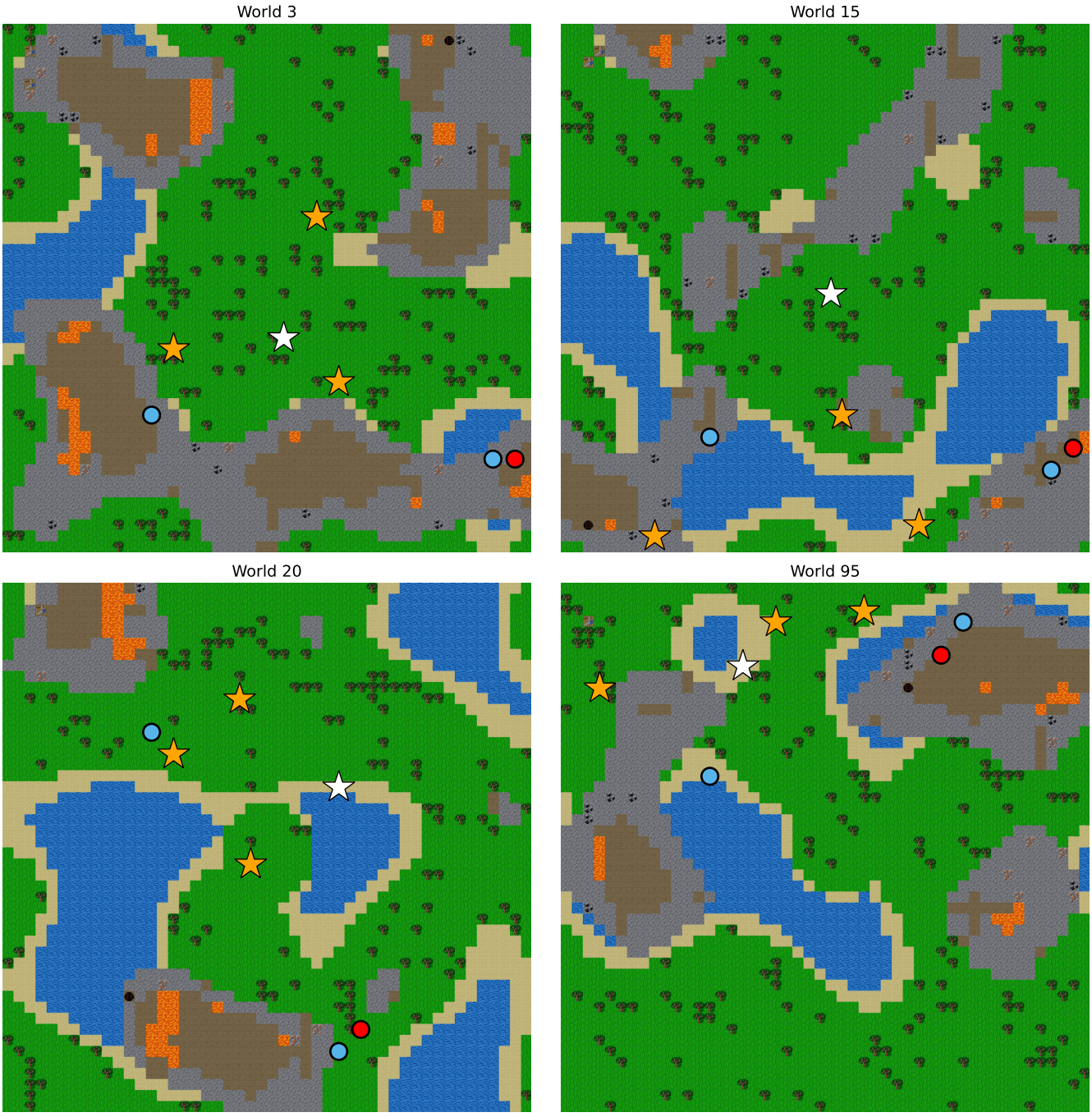
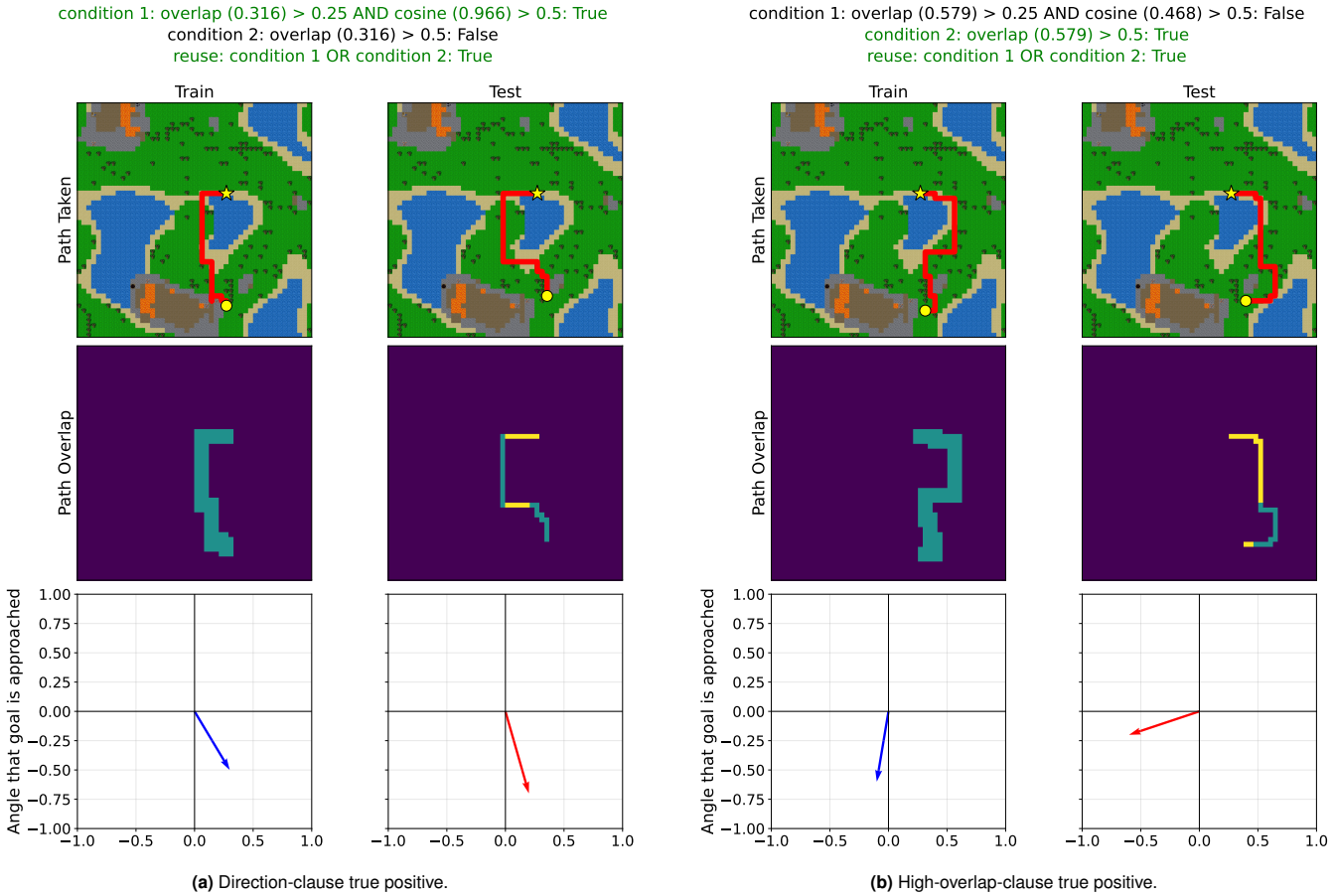


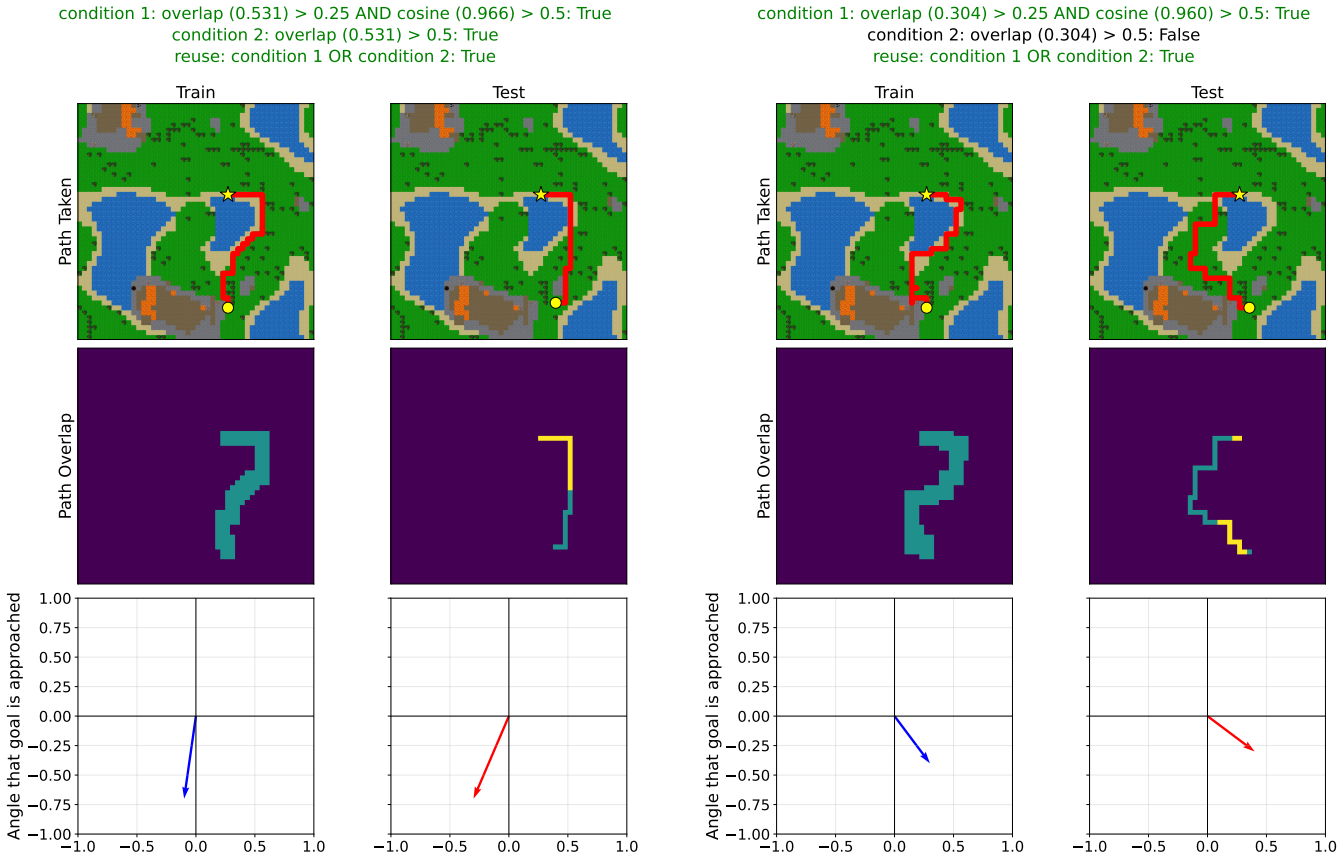
Fig. S19. Craftax environments used in the human experiment (experiment 5). Each panel shows one of the four procedurally generated worlds used across blocks. Stars indicate goal object locations (orange for training objects, white for the test object). Circles indicate possible starting positions (blue and red).



(a) Direction-clause true positive.

(b) High-overlap-clause true positive.

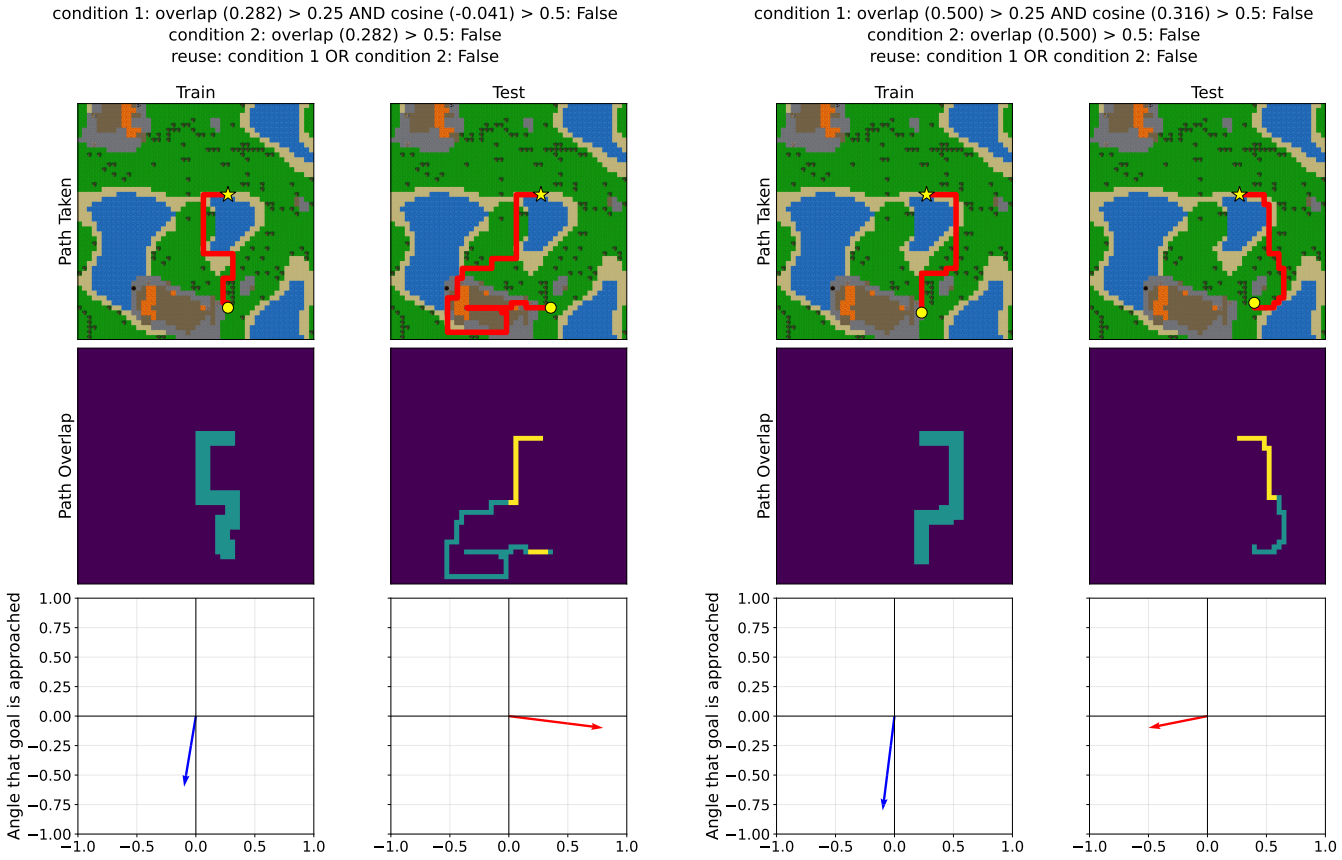
Fig. S20. Craftax path-reuse true positives under each clause. (a) Path overlap is modest ($o_{\text{map}} \approx .316$), but the angle of approach to the test object is consistent across train and test, so the direction clause fires. (b) Approach angles are near-perpendicular across train and test, yet path overlap is large enough that the high-overlap clause fires on its own.



(a) Both-clause true positive.

(b) False positive.

Fig. S21. Craftax path-reuse: both-clause success and a false positive. (a) Both clauses are satisfied: the subject's test path overlaps the training path heavily and approaches the object from the same angle. (b) The criterion fires but the test path does not meaningfully reuse the training path.



(a) True negative.

(b) False negative.

Fig. S22. Craftax path-reuse: true negative and a near-miss false negative. (a) The subject takes a distinct route at evaluation and the criterion correctly returns zero. (b) The subject broadly reuses the training path but falls short on both map overlap and approach angle, so the criterion misses a partial reuse a human observer would credit.

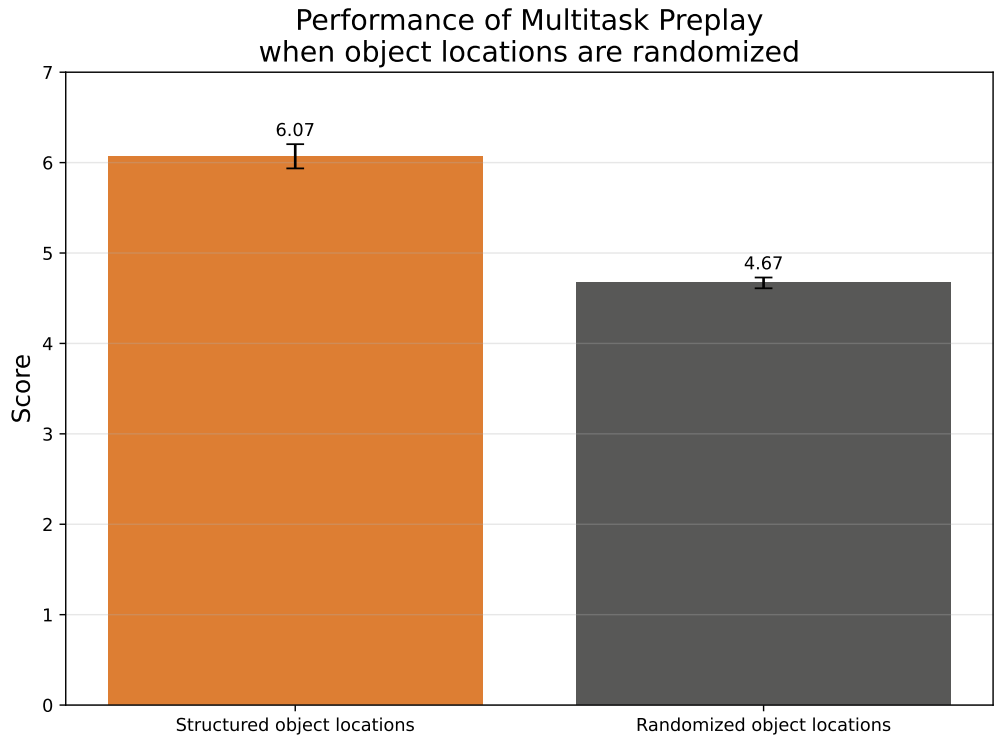


Fig. S23. Performance of Multitask Preplay when object locations are randomized instead of following default object co-occurrence structure.

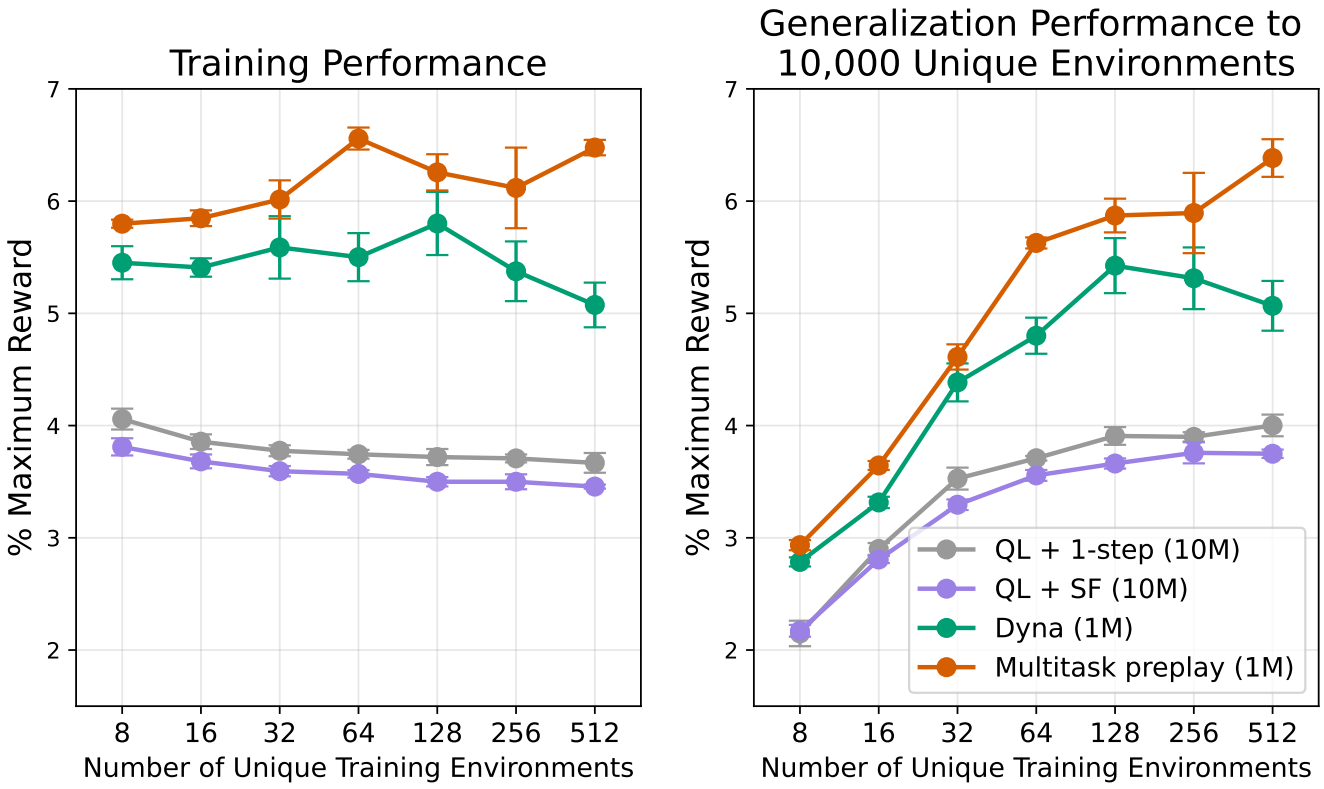


Fig. S24. Multitask Preplay consistently benefits both training and generalization from preplaying offtask goals g'. Training and generalization performance for data in Figure 5.

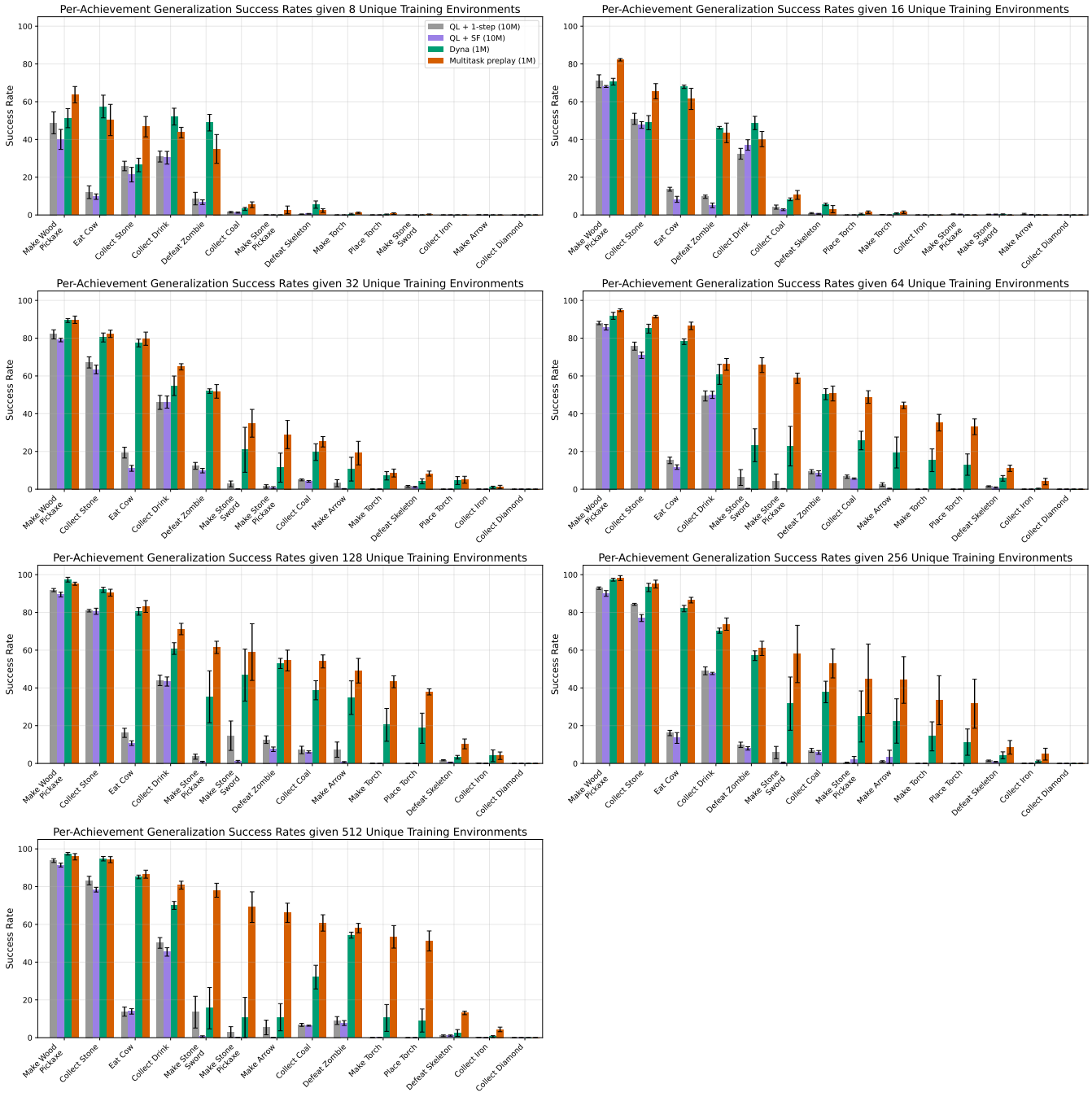


Fig. S25. Generalization performance of Multitask Preplay vs. Dyna. As the number of training environments increases, Multitask Preplay consistently improves its generalization performance, despite learning on fewer samples per environment. In contrast, Dyna, which uses *the exact same simulation budget* as Multitask Preplay, begins to decrease in its generalization performance.

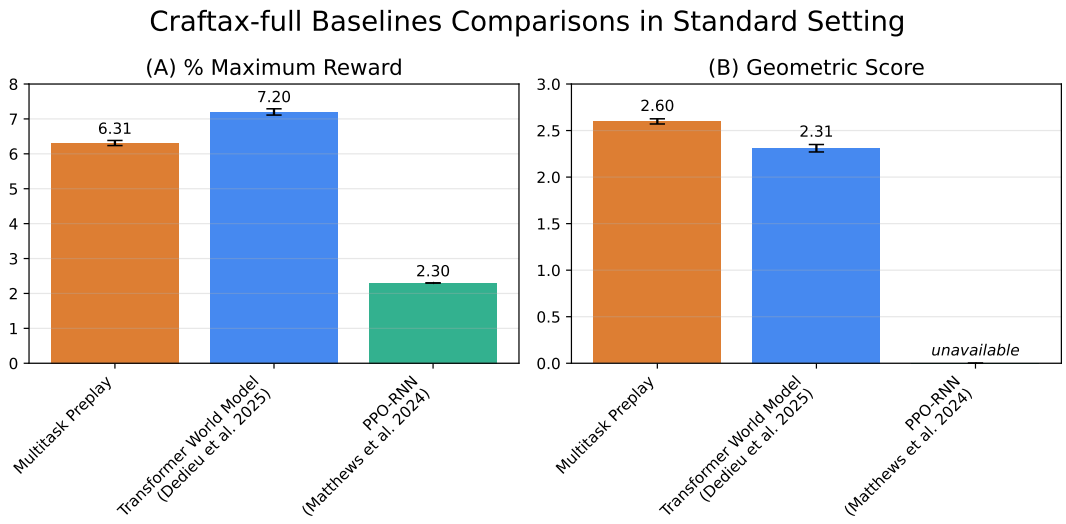


Fig. S26. We compare Multitask Preplay to the state-of-the-art background planning method (Transformer World Model; TWM (18)) and the best model-free method (Proximal Policy Optimization; PPO (19)) in the standard Craftax setting (Craftax-Full (11)). This setting differs from ours in that we study generalization as a function of the number of training environments, whereas the standard setting has a new environment sampled in every episode and measures performance within this episode. Multitask Preplay uses a less expressive architecture for modelling sequential dependencies (an RNN in Multitask Preplay vs. a Transformer (5) in TWM), which may lead to less effective representation learning. We find that it can accomplish a wider range of tasks across episodes (see B), but fewer tasks within an episode (see A). **(A) % Maximum Score** presents the average percentage of all possible tasks completed within an episode. The best background planning methods complete more tasks than the best purely model-free method. We find that TWM completes more tasks within an episode than Multitask Preplay on average. We hypothesize that this is due to its less expressive architecture for modelling sequential dependencies. **(B) Geometric Means** places more weight on occasionally solving many achievements than on consistently solving a subset. It is given by $S = \exp\left(\frac{1}{N}\sum_{i=1}^{N} \ln\left(1+s_i\right)\right) - 1$, where $s_i \in [0, 100]$ is the success percentage for achievement i. This measure was used to measure the performance of TWM but not PPO on Craftax-full. We find that Multitask Preplay has a higher score, indicating that it accomplishes a wider range of tasks. Given that these are generalization scores, we hypothesize that this is due to Multitask Preplay learning about subtasks outside of the ones it experienced during background planning—this in turn could lead it to perform a wider array of tasks at generalization (on average). Future work can look to integrate the more versatile background planning algorithm of Multitask Preplay with the more sophisticated architectural choices of TWM.

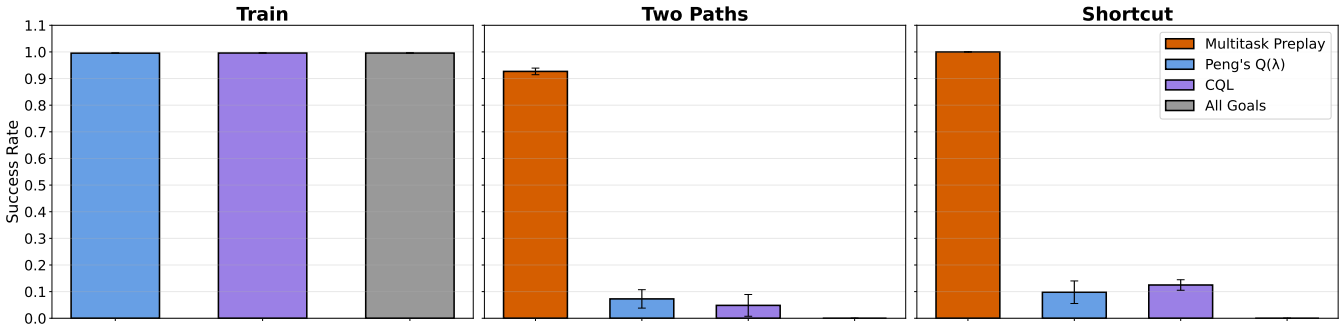


Fig. S27. Ablation of all-goals learning components. Ablation of off-task $Q(\lambda)$, conservative Q-learning (CQL), and the combined conservative all-goals loss.

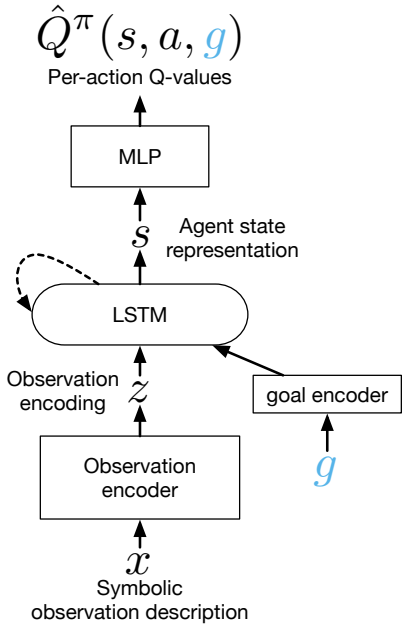


Fig. S28. Diagram of neural network architecture used by all methods that learn Q-values.

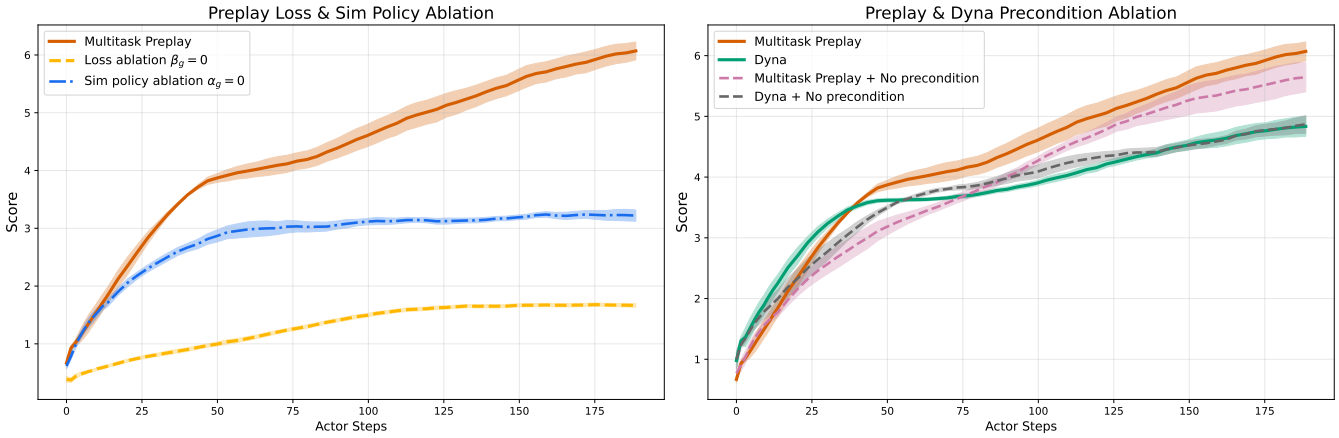


Fig. S29. (Left) Algorithm ablations. It is not only key to simulate towards offtask goals g', but to do so with a small bias towards the main goal g ($\alpha_g > 0$), and to propagate this information to action-value predictions for g (i.e. to have $\beta_g > 0$). Without either, the benefits of counterfactual task simulation are not fully realized. **(Right) Environment ablation**. Multitask preplay benefits from having $\phi_{\texttt{avail}}$ knowledge of when preconditions are met, but this is not required to get the benefits of counterfactual task simulation. Indeed, without this precondition information (Multitask Preplay + No precondition), it still achieves a strong improvement over Dyna.

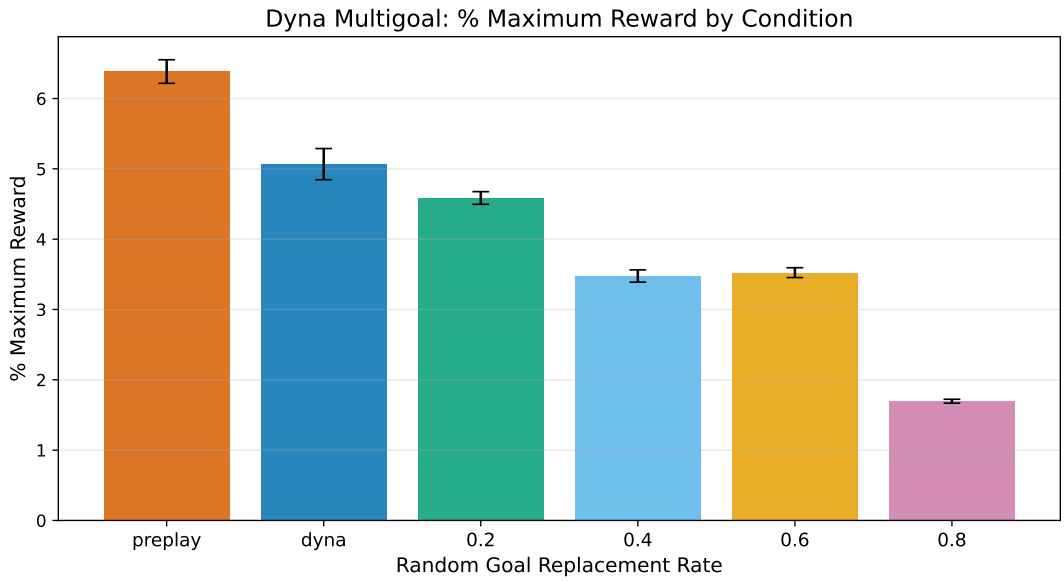


Fig. S30. Dyna with random goal replacement on Craftax. Percentage of maximum reward achieved by Dyna variants that randomly replace the simulation goal with a uniformly sampled goal at varying rates, compared to standard Dyna and Multitask Preplay. Naively introducing off-task simulations does not match the performance of Multitask Preplay's structured all-goals learning.

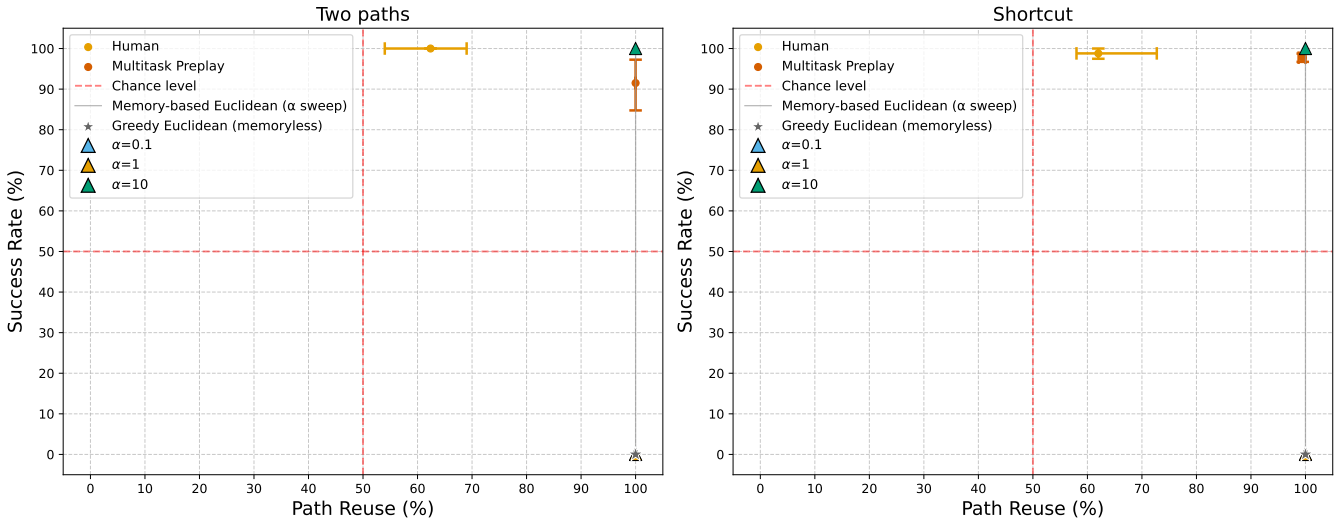


Fig. S31. Euclidean-distance policies in the fully observable JaxMaze domain. Each panel plots success rate against path reuse for the two-paths (Experiment 1, left) and shortcut (Experiment 2, right) tasks. The memory-less greedy Euclidean policy (gray star) achieves near-zero success; minimizing distance to the goal does not by itself solve either task. Adding a bias to reuse previously taken paths (*Memory-based Euclidean Heuristic*; α sweep) changes this—as the path-following weight α grows, both success and path reuse climb, and at $\alpha = 10$ the policy solves both tasks with complete reuse of the training path (success 100%, path reuse 100%; its deterministic policy saturates reuse above the human rate). The main-text figures report the $\alpha = 10$ policy.

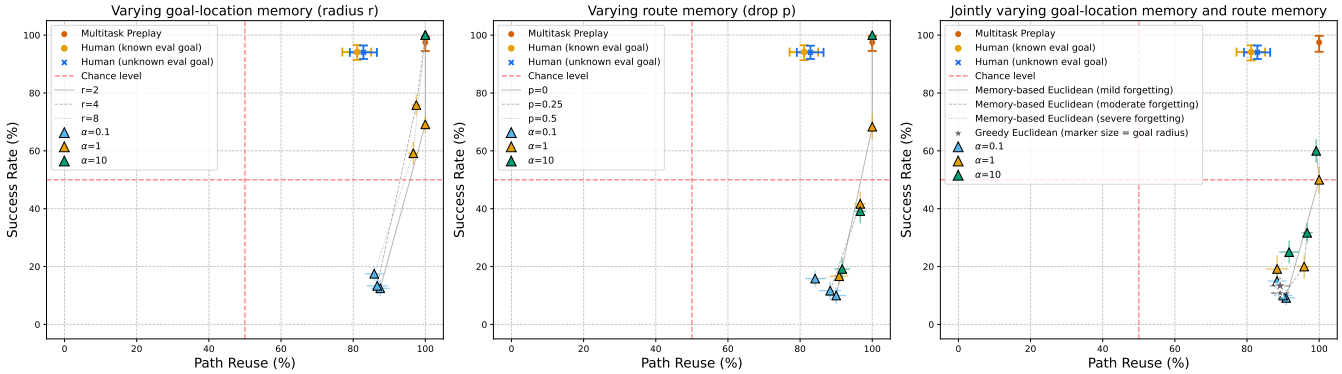


Fig. S32. Euclidean-distance policies under partial observability in Craftax. In the partially observable Craftax domain we model two sources of uncertainty—about the goal location, by sampling the goal within a radius r of its true position, and about the remembered route, by dropping each state in the path memory $M(g)$ with probability p. Taken individually (left, varying r; center, varying p), a strong path-following bias lets the *Memory-based Euclidean Heuristic* tolerate either source of noise and still reach high success and high path reuse. When goal-location memory and route memory are jointly varied at three fidelity levels, mild ($r = 2$, $p = 0.1$), moderate ($r = 4$, $p = 0.3$), and severe ($r = 8$, $p = 0.5$)—levels that pair intermediate values of r and p, so their p grid differs from the center panel's single-source sweep—their effects compound (right); even under mild forgetting the policy tops out near 60% success despite maintaining high path reuse, falling short of human performance. The main-text figures report the $\alpha = 10$ policy under the mild setting.

Table S1. Shared hyperparameters across all models. These parameters are consistent across Q-learning, Successor Features (SF), Dyna, and Multitask Preplay algorithms.

Hyperparameter	Value
Discount factor (γ)	0.992
Replay buffer capacity	50,000
Batch size	32
Gradient clipping	80
Activation function	Leaky ReLU
LSTM hidden size	256
MLP hidden dimensions	1024

Table S2. Model-specific hyperparameters

	Q-learning	SF	HER	Dyna	Multitask Preplay
Number of parallel environments					
JaxMaze	32	32	32	32	32
Craftax	128	128	128	32	32
Batch length					
JaxMaze	40	40	40	40	40
Craftax	80	80	80	40	40
Learning rate					
JaxMaze	10^{-3}	10^{-3}	10^{-4}	3×10^{-4}	3×10^{-4}
Craftax	10^{-3}	10^{-3}	3×10^{-4}	3×10^{-4}	3×10^{-4}
MLP hidden layers					
JaxMaze	2	3	2	2	2
Craftax	3	3	3	2	2
Number of simulations					
JaxMaze	-	-	-	2	2
Craftax (Human)	-	-	-	4	4
Craftax (AI)	-	-	-	10	10
Auxiliary task coefficients					
1-step prediction	-	10^{-5}	-	-	-
SF auxiliary	-	10^{-6}	-	-	-
Simulation length					
JaxMaze	-	-	-	15	15
Craftax	-	-	-	20	20
Conservative Q-learning coefficient					
JaxMaze	-	-	10^{-3}	-	10^{-3}
Craftax (Human)	-	-	10^{-2}	-	10^{-2}
Craftax (AI)	-	-	10^{-3}	-	10^{-2}
Conservative all-goals coefficient					
All settings	-	-	1	-	1
Goal weight (α_g)					
JaxMaze	-	-	-	-	0
Craftax (Human)	-	-	-	-	0
Craftax (AI)	-	-	-	-	1
Off-task goal weight (α'_g)					
JaxMaze	-	-	-	-	1
Craftax (Human)	-	-	-	-	1
Craftax (AI)	-	-	-	-	2

References

1. Kapturowski S, Ostrovski G, Quan J, Munos R, Dabney W (2018) Recurrent experience replay in distributed reinforcement learning in *International conference on learning representations*.
2. Hoffman MW, et al. (2020) Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*.
3. Lu C, et al. (2022) Discovered policy optimisation. *Advances in Neural Information Processing Systems* 35:16455–16468.
4. Mikolov T, Yih Wt, Zweig G (2013) Linguistic regularities in continuous space word representations in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*. pp. 746–751.
5. Vaswani A, et al. (2017) Attention is all you need. *Advances in neural information processing systems* 30.
6. Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural computation* 9(8):1735–1780.

7. Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.
8. Mnih V, et al. (2015) Human-level control through deep reinforcement learning. *nature* 518(7540):529–533.
9. Kingma DP (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
10. Sutton RS, Mahmood AR, Precup D, van Hasselt H (2014) A new $Q(\lambda)$ with interim forward view and Monte Carlo equivalence in *International Conference on Machine Learning*.
11. Matthews M, et al. (2024) Craftax: A lightning-fast benchmark for open-ended reinforcement learning in *International Conference on Machine Learning (ICML)*.
12. Whelan R (2008) Effective analysis of reaction time data. *The psychological record* 58(3):475–482.
13. Ulrich R, Miller J (1993) Information processing models generating lognormally distributed reaction times. *Journal of Mathematical Psychology* 37(4):513–525.
14. Cobbe K, Hesse C, Hilton J, Schulman J (2019) Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*.
15. Küttler H, et al. (2020) The nethack learning environment. *Advances in Neural Information Processing Systems* 33:7671–7684.
16. Kulkarni TD, Narasimhan K, Saeedi A, Tenenbaum J (2016) Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems* 29.
17. Borsa D, et al. (2018) Universal successor features approximators. *arXiv preprint arXiv:1812.07626*.
18. Dedieu A, et al. (2025) Improving transformer world models for data-efficient rl. *ICML*.
19. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.