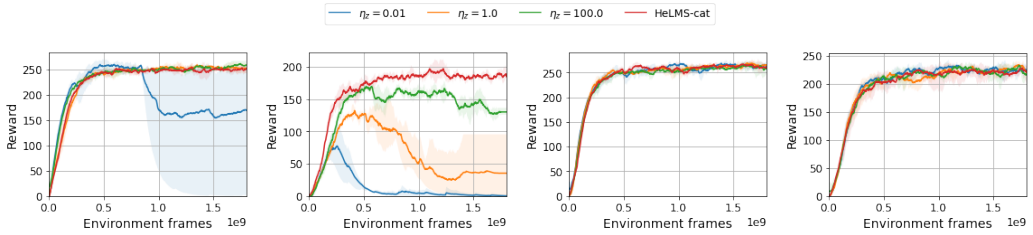
APPENDIX

A ADDITIONAL EXPERIMENTS

A.1 ABLATIONS FOR KL-REGULARISATION

In these experiments, we investigate the effect of KL-regularisation on the mid-level components, both for the offline learning phase (regularising each component to $p(\mathbf{z}_t \mid \mathbf{y}_t) = \mathcal{N}(0, I)$ via coefficient β_z), and the online reinforcement learning stage via HeLMS-mix (regularising each component to the mid-level skills learned offline, via coefficient η_z)). The results are reported in Figure 9, where each plot represents a different setting for offline KL-regularisation (either regularisation to $\mathcal{N}(0, I)$ with $\beta_z = 0.01$, or no regularisation with $\beta_z = 0$) and a different transfer case (the easy case of transferring to object set 4, or the hard case of transferring to object set 3). Each plot shows the downstream performance when varying the strength of KL-regularisation during RL via coefficient η_z. The HeLMS-cat approach represents the extreme case where the skills are entirely frozen (i.e. full regularisation).

The results suggest some interesting properties of the latent skill space based on regularisation. When regularising the mid-level components to the $\mathcal{N}(0, I)$ prior, it is important to regularise during online RL; this is especially true for the hard transfer case, where HeLMS-cat performs much better, and the performance degrades significantly with lower regularisation values. However, when removing mid-level regularisation during offline learning, the method is insensitive to regularisation during RL over the entire range evaluated, from 0.01 to 100.0. We conjecture that with mid-level skills regularised to $\mathcal{N}(0, I)$, the different mid-level skills are drawn closer together and occupy a more compact region in latent space, such that KL-regularisation is necessary during RL for a skill to avoid drifting and overlapping with the latent distribution of other skills (i.e. skill degeneracy). In contrast, without offline KL-regularisation, the skills are free to expand and occupy more distant regions of the latent space, rendering further regularisation unnecessary during RL. Such latent space properties could be further analysed to improve learning and transfer of skills; we leave this as an interesting direction for future work.



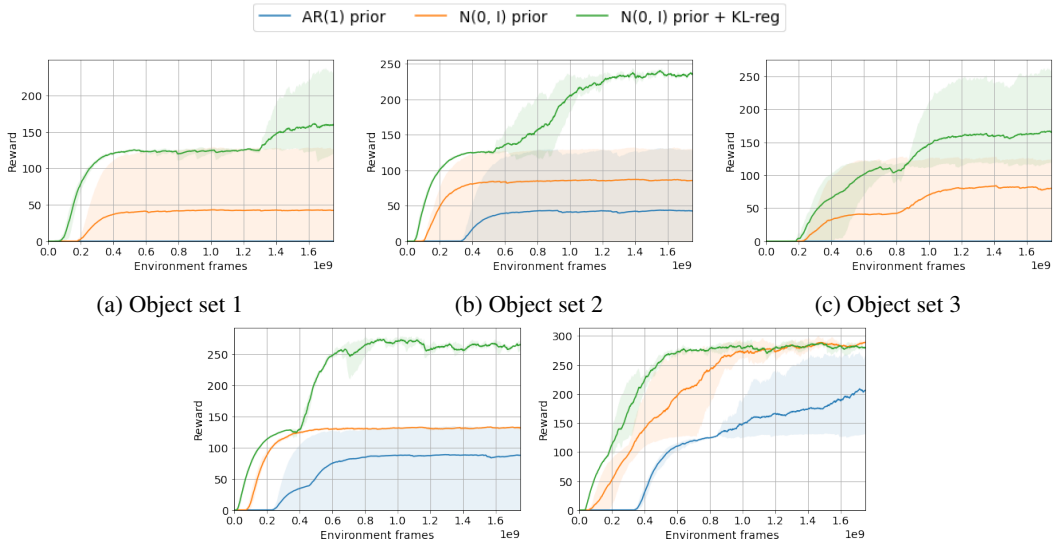
(a) $\beta_z = 0.01$, easy (b) $\beta_z = 0.01$, hard (c) $\beta_z = 0$, easy (d) $\beta_z = 0$, hard

Figure 9: Ablations for KL-regularisation, showing downstream performance with different degrees of online KL-regularisation. Performance is evaluated for easy (object set 4) and hard (object set 3) transfer cases, with sparse staged rewards; when using different offline regularisation coefficient (β_z) values for the mid-level components.

A.2 NPMP ABLATION

The Neural Probabilistic Motor Primitives (NPMP) work (Merel et al., 2019) presents a strong baseline approach to learning transferable motor behaviours, and we run ablations to ensure a fair comparison to the strongest possible result. As discussed in the main text, NPMP employs a Gaussian high-level latent encoder with a $AR(1)$ prior in the latent space. We also try a fixed $N(0, I)$ prior (this is equivalent to an $AR(1)$ prior with a coefficient of 0, so can be considered a hyperparameter choice). Since our method benefits from KL-regularisation during RL, we apply this to NPMP as well.

As shown in Figure 10, we find that both changes lead to substantial improvements in the manipulation domain, on all five object sets. Consequently, in our main experiments, we report results with the best variant, using a $N(0, I)$ prior with KL-regularisation during RL.



(a) Object set 1 (b) Object set 2 (c) Object set 3

(d) Object set 4 (e) Object set 5

Figure 10: Ablation for NPMP (Merel et al., 2019) using staged sparse reward, with all object sets. We find that using $N(0, I)$ prior with KL-regularisation performs much better in the manipulation domain compared to the original $AR(1)$ prior, so we use this modified baseline.

B Reinforcement learning with MPO and RHPO

As discussed in Section 3.2, the hierarchy of skills are transferred to RL in two ways: HeLMS-cat, which learns a new high-level categorical policy $\pi(\mathbf{y}_t \,|\, \mathbf{x}_t)$ via MPO (Abdolmaleki et al., 2018); or HeLMS-mix, which learns a mixture policy $\pi(\mathbf{z}_t \,|\, \mathbf{x}_t) = \sum_{\mathbf{y}_t} \pi(\mathbf{y}_t \,|\, \mathbf{x}_t)\pi(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t)$ via RHPO (Wulfmeier et al., 2020). We describe the optimisation for both of these cases in the following subsections. For clarity of notation, we omit the additional KL-regularisation terms introduced in Section 3.2 and describe just the base methods of MPO and RHPO when applied to the RL setting in this paper. These KL-terms are incorporated as additional loss terms in the policy improvement stage.

B.1 HeLMS-cat via MPO

Maximum a posteriori Policy Optimisation (MPO) is an Expectation-Maximisation-based algorithm that performs off-policy updates in three steps: (1) updating the critic; (2) creating a non-parametric intermediate policy by weighting sampled actions using the critic; and (3) updating the parametric policy to fit the critic-reweighted non-parametric policy, with trust region constraints to improve stability. We detail each of these steps below. Note that while the original MPO operates in the environment's action space, we use it here for the high-level controller, to set the categorical variable $\mathbf{y}_t$.

Policy evaluation First, the critic is updated via a TD(0) objective as:

$$\min_{\theta} L(\theta) = \mathbb{E}_{\mathbf{x}_t, \mathbf{y}_t \sim \mathcal{B}} \Big[\big(Q_{\mathrm{T}} - Q_\phi(\mathbf{x}_t, \mathbf{y}_t) \big)^2 \Big], \tag{6}$$

Here, $Q_{\mathrm{T}} = r_t + \gamma \mathbb{E}_{\mathbf{x}_{t+1}, \mathbf{y}_{t+1}}[Q'(s_{t+1}, \mathbf{y}_{t+1})]$ is the 1-step target with the state transition $(\mathbf{x}_t, \mathbf{y}_t, \mathbf{x}_{t+1})$ returned from the replay buffer $\mathcal{B}$, and next action sampled from $\mathbf{y}_{t+1} \sim \pi'(\cdot|\mathbf{x}_{t+1})$. π' and Q' are target networks for the policy and the critic, used to stabilise learning.

Policy improvement Next, we proceed with the first step of policy improvement by constructing an intermediate non-parametric policy $q(\mathbf{y}_t|\mathbf{x}_t)$, and optimising the following constrained objective:

$$\max_q J(q) = \mathbb{E}_{\mathbf{y}_t\sim q, \mathbf{x}_t\sim\mathcal{B}}\big[Q_\phi(\mathbf{x}_t, \mathbf{y}_t)\big], \;\; \text{s.t. } \mathbb{E}_{\mathbf{x}_t\sim\mathcal{B}}\Big[\mathrm{KL}\big(q(\cdot|\mathbf{x}_t)\|\pi_{\theta_k}(\cdot|\mathbf{x}_t)\big)\Big] \leq \epsilon_E, \tag{7}$$

where ϵ_E defines a bound on the KL divergence between the non-parametric and parametric policies at the current learning step k. This constrained optimisation problem has the following closed-form solution:

$$q(\mathbf{y}_t \,|\, \mathbf{x}_t) \propto \pi_{\theta_k}(\mathbf{y}_t \,|\, \mathbf{x}_t) \exp\left(Q_\phi(\mathbf{x}_t, \mathbf{y}_t)/\eta\right). \tag{8}$$

In other words, this step constructs an intermediate policy which reweights samples from the previous policy using exponentiated temperature-scaled critic values. The temperature parameter η is derived based on the dual of the Lagrangian; for futher details please refer to (Abdolmaleki et al., 2018).

Finally, we can fit a parametric policy to the non-parametric distribution $q(\mathbf{y}_t \,|\, \mathbf{x}_t)$ by minimising their KL-divergence, subject to a trust-region constraint on the parametric policy:

$$\begin{aligned}\theta_{k+1} = \arg\min_\theta \mathbb{E}_{\mathbf{x}_t\sim\mathcal{B}}\Big[\mathrm{KL}(q(\mathbf{y}_t \,|\, \mathbf{x}_t) \,||\, \pi_\theta(\mathbf{y}_t|\mathbf{x}_t))\Big], \\ \text{s.t. } \mathbb{E}_{\mathbf{x}_t\sim\mathcal{B}}\Big[\mathrm{KL}\big(\pi_{\theta_{k+1}}(\mathbf{y}_t \,|\, \mathbf{x}_t) \,||\, \pi_{\theta_k}(\mathbf{y}_t \,|\, \mathbf{x}_t)\big)\Big] \leq \epsilon_M.\end{aligned} \tag{9}$$

This optimisation problem can be solved via Lagrangian relaxation, with the Lagrangian multiplier ϵ_M modulating the strength of the trust-region constraint. For further details and full derivations, please refer to (Abdolmaleki et al., 2018).

B.2 HELMS-MIX VIA RHPO

RHPO (Wulfmeier et al., 2020) follows a similar optimisation procedure as MPO, but extends it to mixture policies and multi-task settings. We do not exploit the multi-task capability in this work, but utilise RHPO to optimise the mixture policy in latent space, $\pi(\mathbf{z}_t \,|\, \mathbf{x}_t) = \sum_{\mathbf{y}_t} \pi(\mathbf{y}_t \,|\, \mathbf{x}_t)\pi(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t)$. The Q-function $Q_\phi(\mathbf{x}_t, \mathbf{z}_t)$ and parametric policy $\pi_{\theta_k}(\mathbf{z}_t \,|\, \mathbf{x}_t)$ use the continuous latents $\mathbf{z}_t$ as actions instead of the categorical $\mathbf{y}_t$. This is also in contrast to the original formulation of RHPO, which uses the environment's action space. Compared to MPO, the policy improvement stage of the non-parametric policy is minimally adapted to take into account the new mixture policy. The key difference is in the parametric policy update step, which optimises the following:

$$\begin{aligned}\theta_{k+1} = \arg\min_\theta \; \mathbb{E}_{\mathbf{x}_t\sim\mathcal{B}}\Big[\mathrm{KL}(q(\mathbf{z}_t \,|\, \mathbf{x}_t) \,||\, \pi_\theta(\mathbf{z}_t|\mathbf{x}_t))\Big], \\ \text{s.t. } \mathbb{E}_{\mathbf{x}_t\sim\mathcal{B}}\Big[\mathrm{KL}\big(\pi_{\theta_{k+1}}(\mathbf{y}_t \,|\, \mathbf{x}_t) \,||\, \pi_{\theta_k}(\mathbf{y}_t \,|\, \mathbf{x}_t)\big) \\ + \sum_{\mathbf{y}_t} \mathrm{KL}\big(\pi_{\theta_{k+1}}(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t) \,||\, \pi_{\theta_k}(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t)\big)\Big] \leq \epsilon_M.\end{aligned} \tag{10}$$

In other words, separate trust-region constraints are applied to a sum of KL-divergences: for the high-level categorical and for each of the mixture components. Following the original RHPO, we separate the single constraint into decoupled constraints that set a different ϵ for the means, covariances, and categorical (ϵ_μ, ϵ_σ, and ϵ_{cat}, respectively). This allows the optimiser to independently modulate how much the categorical distribution, component means, and component variances can change. For further details and full derivations, please refer to (Wulfmeier et al., 2020).

C ELBO DERIVATION AND INTUITIONS

We can compute the Evidence Lower Bound for the state-conditional action distribution, $p(\mathbf{a}_{1:T} \,|\, \mathbf{x}_{1:T}) \geq ELBO$, as follows:

$$\begin{aligned}
ELBO &= p(\mathbf{a}_{1:T} \,|\, \mathbf{x}_{1:T}) - \mathrm{KL}(q(\mathbf{y}_{0:T}, \mathbf{z}_{1:T} \,|\, \mathbf{x}_{1:T}) \,||\, p(\mathbf{y}_{0:T}, \mathbf{z}_{1:T} \,|\, \mathbf{x}_{1:T})) \\
&= \mathbb{E}_{q(\mathbf{y}_{0:T}, \mathbf{z}_{1:T} \,|\, \mathbf{x}_{1:T})} \Big[\log p(\mathbf{a}_{1:T}, \mathbf{y}_{0:T}, \mathbf{z}_{1:T} \,|\, \mathbf{x}_{1:T}) - \log q(\mathbf{y}_{0:T}, \mathbf{z}_{1:T} \,|\, \mathbf{x}_{1:T}) \Big] \\
&= \mathbb{E}_{q_{1:T}} \Bigg[\sum_{t=1}^{T} \log p(\mathbf{a}_t \,|\, \mathbf{z}_t, \mathbf{x}_t) + \log p(\mathbf{z}_t \,|\, \mathbf{y}_t) + \log p(\mathbf{y}_t \,|\, \mathbf{y}_{t-1}) \\
&\quad - \log q(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t) - \log q(\mathbf{y}_t \,|\, \mathbf{y}_{t-1}, \mathbf{x}) \Bigg] \\
&= \sum_{t=1}^{T} \mathbb{E}_{q_{1:T}} \Bigg[\log p(\mathbf{a}_t \,|\, \mathbf{z}_t, \mathbf{x}_t) - \mathrm{KL}(q(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t) \,||\, p(\mathbf{z}_t \,|\, \mathbf{y}_t)) \\
&\quad - \mathrm{KL}(q(\mathbf{y}_t \,|\, \mathbf{y}_{t-1}, \mathbf{x}_t) \,||\, p(\mathbf{y}_t \,|\, \mathbf{y}_{t-1})) \Bigg] \qquad (11)
\end{aligned}$$

We note that the first two terms in the expectation depend only on timestep t, so we can simplify and marginalise exactly over all discrete $\{\mathbf{y}_{1:T}\} \backslash \mathbf{y}_t$. For the final term, we note that the KL at timestep t is constant with respect to $\mathbf{y}_t$ (as it already marginalises over the whole distribution), and only depends on $\mathbf{y}_{t-1}$. Lastly, we will use sampling to approximate the expectation over $\mathbf{z}_t$. This yields the following:

$$\begin{aligned}
ELBO &= \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t)} \Bigg[\sum_{\mathbf{y}_{0:T}} q(\mathbf{y}_{0:T} \,|\, \mathbf{x}_{1:T}) \Big(\log p(\mathbf{a}_t \,|\, \mathbf{z}_t, \mathbf{x}_t) - \mathrm{KL}(q(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t) \,||\, p(\mathbf{z}_t \,|\, \mathbf{y}_t)) \\
&\quad - \mathrm{KL}(q(\mathbf{y}_t \,|\, \mathbf{y}_{t-1}, \mathbf{x}_t) \,||\, p(\mathbf{y}_t \,|\, \mathbf{y}_{t-1})) \Big) \Bigg] \\
ELBO &\approx \sum_{t=1}^{T} \Bigg[\sum_{\mathbf{y}_t} q(\mathbf{y}_t \,|\, \mathbf{x}_{1:t}) \Big(\overbrace{\log p(\mathbf{a}_t \,|\, \tilde{\mathbf{z}}_t^{\{\mathbf{y}_t\}}, \mathbf{x}_t)}^{\text{per-component recon loss}} - \beta_z \overbrace{\mathrm{KL}(q(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t) \,||\, p(\mathbf{z}_t \,|\, \mathbf{y}_t))}^{\text{per-component KL regulariser}} \Big) \Bigg] \\
&\quad - \beta_y \sum_{t=1}^{T} \Bigg[\sum_{\mathbf{y}_{t-1}} q(\mathbf{y}_{t-1} \,|\, \mathbf{x}_{1:t-1}) \underbrace{\mathrm{KL}(q(\mathbf{y}_t \,|\, \mathbf{y}_{t-1}, \mathbf{x}_t) \,||\, p(\mathbf{y}_t \,|\, \mathbf{y}_{t-1}))}_{\text{discrete regulariser}} \Bigg] \qquad (12)
\end{aligned}$$

where $\tilde{\mathbf{z}}_t^{\{\mathbf{y}_t\}} \sim q(\mathbf{z}_t \,|\, \mathbf{y}_t, \mathbf{x}_t)$, the coefficients β_y and β_z can be used to weight the KL terms, and the cumulative component probability $q(\mathbf{y}_t \,|\, \mathbf{x}_{1:t})$ can be computed iteratively as:

$$q(\mathbf{y}_t \,|\, \mathbf{x}_{1:t}) = \sum_{\mathbf{y}_{t-1}} q(\mathbf{y}_t \,|\, \mathbf{y}_{t-1}, \mathbf{x}_t) q(\mathbf{y}_{t-1} \,|\, \mathbf{x}_{1:t-1}) \qquad (13)$$

In other words, for each timestep t and each mixture component, we compute the latent sample and the corresponding action log-probability, and the KL-divergence between the component posterior and prior. This is then marginalised over all $\mathbf{y}_t$, with an additional KL over the categorical transitions.

Structuring the graphical model and ELBO in this form has a number of useful properties. First, the ELBO terms include an action reconstruction loss and KL term for each mixture component, scaled by the posterior probability of each component given the history. For a given state, this pressures the model to assign higher posterior probability to components that have low reconstruction cost or KL, which allows different components to specialise for different parts of the state space. Second, the categorical KL between posterior and prior categorical transition distributions is scaled by the

posterior probability of the previous component given history $q(\mathbf{y}_{t-1} \mid \mathbf{x}_{1:t-1})$: this allows the relative probabilities of past skill transitions along a trajectory to be considered when regularising the current skill distribution. Finally, this formulation does not require any sampling or backpropagation through the categorical variable: starting from $t = 0$, the terms for each timestep can be efficiently computed by recursively updating the posterior over components given history ($q(\mathbf{y}_t \mid \mathbf{x}_{1:t})$), and summing over all possible categorical values at each timestep.

D ENVIRONMENT PARAMETERS

As discussed earlier in the paper, all experiments take place in a MuJoCo-based object manipulation environment using a Sawyer robot manipulator and three objects: red, green, and blue. The state variables in the Sawyer environment are shown in Table 3. All state variables are stacked for 3 frames for all agents. The object states are only provided to the mid-level and high-level for HeLMS runs, and the camera images are only used by the high- and mid-level controller in the vision transfer experiments (without object states).

The action space is also shown in Table 4. Since the action dimensions vary significantly in range, they are normalised to be between $[-1, 1]$ for all methods during learning.

When learning via RL, we apply domain randomisation to physics (but not visual randomisation), and a randomly sampled action delay of 0-2 timesteps. This is applied for all approaches, and ensures that we can learn a policy that is robust to small changes in the environment.

Proprioception

State	Dims
Joint angles	7
Joint velocities	7
Joint torque	7
TCP pose	7
TCP velocity	6
Wrist angle	1
Wrist velocity	1
Wrist force	3
Wrist torque	3
Binary grasp sensor	1

Object states

State	Dims
Absolute pose (red)	7
Absolute pose (green)	7
Absolute pose (blue)	7
Distance to pinch (red)	7
Distance to pinch (green)	7
Distance to pinch (blue)	7

Vision

State	Dims
Camera images	$64 \times 64 \times 3$

Table 3: Details of state variables used by the agent.

Action	Dims	Range
Gripper translational velocity (x-y-z)	3	$[-0.07, 0.07]$ m/s
Wrist rotation velocity	1	$[-1, 1]$ rad/s
Finger speed	1	$[-255, 255]$ tics/s

Table 4: Action space details for the Sawyer environment.

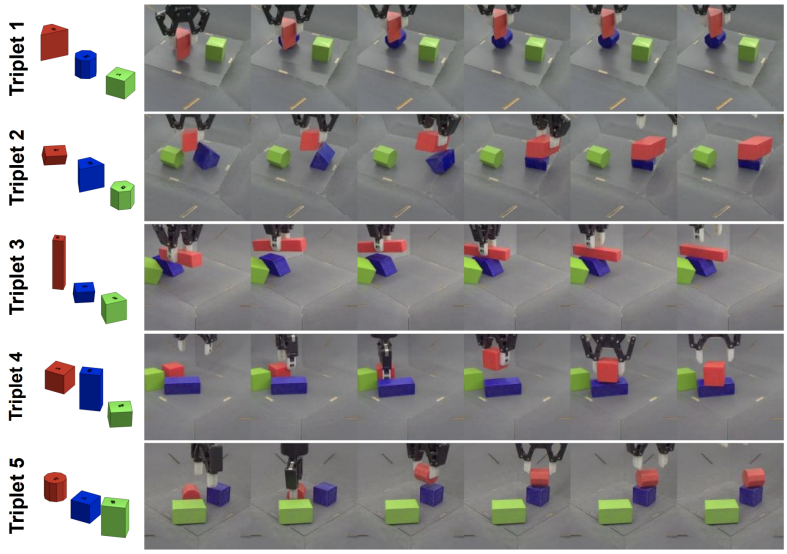


Figure 11: The five object sets (triplets) used in the paper. This image has been taken directly from (Lee et al., 2021) for clarity.

D.1 Object Sets

As discussed in the main paper, we use the object sets defined by Lee et al. (2021), which are carefully designed to cover different object geometries and affordances, presenting different challenges for object interaction tasks. The object sets are shown in Figure 11 (the image has been taken directly from (Lee et al., 2021) for clarity), and feature both simulated and real-world versions; in this paper we focus on the simulated versions. As discussed in detail by (Lee et al., 2021), each object set has a different degree of difficulty and presents a different challenge to the task of stacking red-on-blue:

- In object set 1, the red object has slanted surfaces that make it difficult to grasp, while the blue object is an octagonal prism that can roll.
- In object set 2, the blue object has slanted surfaces, such that the red object will likely slide off unless the blue object is first reoriented.
- In object set 3, the red object is long and narrow, requiring a precise grasp and careful placement.
- Object set 4 is the easiest case with rectangular prisms for both red and blue.
- Object set 5 is also relatively easy, but the blue object has ten faces, meaning limited surface area for stacking.

For more details about the object sets and the rationale behind their design, we refer the reader to (Lee et al., 2021).

E Network Architectures and Hyperparameters

The network architecture details and hyperparameters for HeLMS are shown in Table 5. Parameter sweeps were performed for the β coefficients during offline learning and the η coefficients during RL. Small sweeps were also performed for the RHPO ϵ parameters (refer to (Wulfmeier et al., 2020) for details), but these were found to be fairly insensitive. All other parameters were kept fixed, and used for all methods except where highlighted in the following subsections. All RL experiments were run with 3 seeds to capture variation in each method.

For network architectures, all experiments except for vision used simple 2-layer MLPs for the high- and low-level controllers, and for each mid-level mixture component. An input representation network was used to encode the inputs before passing them to the networks that were learned from scratch: i.e. the high-level for state-based experiments, and both high- and mid-level for vision (re-

call that while the state-based experiments can reuse the mid-level components conditioned on object state, the vision-based policy learned them from scratch and KL-regularised to the offline mid-level skills). The critic network was a 3-layer MLP, applied to the output of another input representation network (separate to the actor, but with the same architecture) with concatenated action.

Offline learning parameters

Name	Value
Latent space dimension	8
Number of mid-level components, K	5 for `red_on_blue` data, 10 for `all_pairs` data
Low-level network	2-hidden layer MLP, $\{256, 256\}$ units
Low-level head	Gaussian, tanh-on-mean, fixed $\sigma = 0.1$
Mid-level network	2-hidden layer MLP for each component, $\{256, 256\}$ units
Mid-level head	Gaussian, learned $\sigma \in [0.01, 1.0]$
High-level network	2-hidden layer MLP, $\{256, 256\}$ units
High-level head	K-way softmax
Activation function	elu
Encoder look-ahead duration	5 timesteps
β_y	1.0
β_z	0.1, 0.0 (object generalisation)
Batch size	128
Learning rate	10^{-4}
Dataset trajectory length	25

Online RL parameters

Name	Value
Number of seeds	3 (all experiments)
Input representation network (state)	Input normalizer layer (linear layer with 256 units, layer-norm, and tanh-on-output)
High-level network (state)	2-hidden layer MLP, $\{256, 256\}$ units
Input representation network (vision)	MLP on proprio and ResNet with three layers of $\{2, 2, 2\}$ blocks corresponding to $\{32, 64, 128\}$ channels
High-level network (vision)	2-hidden layer MLP, $\{256, 256\}$ units
Mid-level network (vision)	2-hidden layer MLP for each component, $\{256, 256\}$ units
Critic network	3-hidden layer MLP, $\{256, 256, 256\}$ units with RNN
Activation function	elu
η_y	0.1 (vision), 0.01
η_z	0.1 (pyramid and vision), 0.01 (object generalisation)
Number of actors	1500
Batch size	512
Trajectory length	10
Learning rate	2×10^{-4}
Number of action samples	20
RHPO categorical constraint ϵ_{cat}	1.0
RHPO mean constraint ϵ_μ	5×10^{-3}
RHPO covariance constraint ϵ_σ	10^{-4}

Table 5: Hyperparameters and architecture details for HeLMS, for both offline training and RL.

F REWARDS

Throughout the experiments, we employ different reward functions for different tasks and to study the efficacy of our method in sparse versus dense reward scenarios.

Reward stages and primitive functions The reward functions for stacking and pyramid tasks use various reward primitives and staged rewards for completing sub-tasks. Each of these rewards are within the range of $[0, 1]$

These include:

- `reach(obj)`: a shaped distance reward to bring the TCP to within a certain tolerance of `obj`.
- `grasp()`: a binary reward for triggering the gripper's grasp sensor.
- `close_fingers()`: a shaped distance reward to bring the fingers inwards.
- `lift(obj)`: shaped reward for lifting the gripper sufficiently high above `obj`.
- `hover(obj1,obj2)`: shaped reward for holding `obj1` above `obj2`.
- `stack(obj1,obj2)`: a sparse reward, only provided if `obj1` is on top of `obj2` to within both a horizontal and vertical tolerance.
- `above(obj,dist)`: shaped reward for being `dist` above `obj`, but anywhere horizontally.
- `pyramid(obj1,obj2,obj3)`: a sparse reward, only provided if `obj3` is on top of the point midway between `obj1` and `obj2`, to within both a horizontal and vertical tolerance.
- `place_near(obj1,obj2)`: sparse reward provided if `obj1` is sufficiently near `obj2`.

Dense stacking reward The dense stacking reward contains a number of stages, where each stage represents a sub-task and has a maximum reward of 1. The stages are:

- `reach(red) AND grasp()`: Reach and grasp the red object.
- `lift(red) AND grasp()`: Lift the red object.
- `hover(red,blue)`: Hover with the red object above the blue object.
- `stack(red,blue)`: Place the red object on top of the blue one.
- `stack(red,blue) AND above(red)`: Move the gripper above after a completed stack.

At each timestep, the latest stage to receive non-zero reward is considered to be the current stage, and all previous stages are assigned a reward of 1. The reward for this timestep is then obtained by summing rewards for all stages, and scaling by the number of stages, to ensure the highest possible reward on any timestep is 1.

Sparse staged stacking reward The sparse staged stacking reward is similar to the dense reward variant, but each stage is sparsified by only providing the reward for the stage once it exceeds a value of 0.95.

This scenario emulates an important real-world problem: that it may be difficult in certain cases to specify carefully shaped meaningful rewards, and it can often be easier to specify (sparsely) whether a condition (such as stacking) has been met.

Sparse stacking reward This fully sparse reward uses the `stack(red,blue)` function to provide reward only when conditions for stacking red on blue have been met.

Pyramid reward The pyramid-building reward uses a staged sparse reward, where each stage represents a sub-task and has a maximum reward of 1. If a stage has dense reward, it is sparsified by only providing the reward once it exceeds a value of 0.95. The stages are:

- `reach(red) AND grasp()`: Reach and grasp the red object.
- `lift(red) AND grasp()`: Lift the red object.
- `hover(red,green)`: Hover with the red object above the green object (with a larger horizontal tolerance, as it does not need to be directly above).
- `place_near(red,green)`: Place the red object sufficiently close to the green object.
- `reach(blue) AND grasp()`: Reach and grasp the blue object.
- `lift(blue) AND grasp()`: Lift the blue object.
- `hover(blue,green) AND hover(blue,red)`: Hover with the blue object above the central position between red and green objects.
- `pyramid(blue,red,green)`: Place the blue object on top to make a pyramid.
- `pyramid(blue,red,green) AND above(blue)`: Move the gripper above after a completed stack.

At each timestep, the latest stage to receive non-zero reward is considered to be the current stage, and all previous stages are assigned a reward of 1. The reward for this timestep is then obtained by summing rewards for all stages, and scaling by the number of stages, to ensure the highest possible reward on any timestep is 1