A Learning and Generalization Challenges

Our experimental setup for §5.1 follows the same setup as Barreto et al. (2018). For n tasks with reward functions $\{r_i\}_{i=1}^n$, they showed that discovered features which predict rewards, i.e. $[\tilde{\phi}_1, \ldots, \tilde{\phi}_n] = [\tilde{r}_1, \ldots, \tilde{r}_n]$, were effective features for transfer with SF&GPI. In their experimental setup, training tasks were one-hot vectors, so regressing $||r - \tilde{\phi}^\top w||$ effectively led ϕ_i to be a feature that predicted reward r_i. We use this same setup, where now *individual modules* are used for both $\tilde{\phi}^{(i)}$ and $\tilde{\psi}^{(i)}$.

While this is a simple method for learning features, it some difficulties. First, there is an imbalance of rewarding vs. non-rewarding transitions, which leads to challenges in learning ϕ. In our setting, an *optimal policy* only has 7% of its transitions be rewarding transitions. Over the course of training, the percentage is much lower. Second, Barreto et al. (2018) were mainly able to show *continual learning* results with additional learning using their discovered features and had **limited success with zero-shot transfer**.

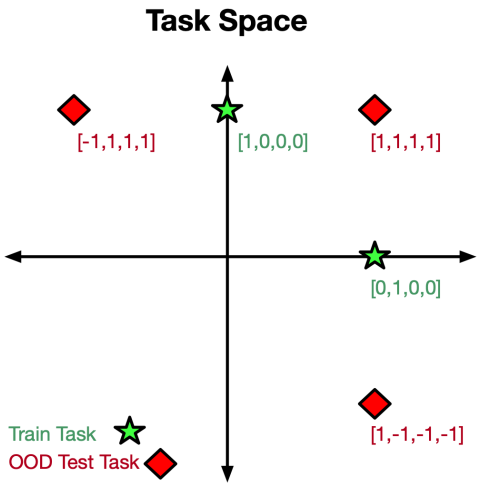

Figure 8: We present a diagram providing intuition for the type of generalization challenge required for our tasks. Green stars represent our training tasks. Red diamonds represent test tasks. They are both outside of training distribution for task encodings and can be far in task-space from training task encodings.

Zero-shot transfer is challenges because training tasks are "out-of-distribution" while additionally requiring behaviors that the agent was never trained on. By out-of-distribution, we mean that the agent is only trained on basis vectors (see Figure 8). However, at test time, the agent must perform tasks that are are far in "task-space" from the training distribution. For example, $w_{\texttt{test}} = [-1, 1, -1, -1]$ has an L2 distance of 1.73 from closest training task. This requires an agent that can generalize to test reward functions that are quite different from training reward functions. While both USFA-Learned-ϕ and MSFA get about the same training error for predicting rewards, we see that MSFA gets a dramatically better reward prediction error for test reward functions $\tilde{r}_{\texttt{test}} = \tilde{\phi}^\top w_{\texttt{test}}$.

Reward prediction error on generalization tasks. To investigate this, we collect 40 episodes of each generalization task and compute the reward prediction error of MSFA and USFA-Learned-ϕ for each task. We present the results in Table 1.

Task	MSFA Error	USFA-Learned-ϕ Error
1,1,0,0	0.0003 ± 0.0090	0.1362 ± 0.3410
1,1,.5,.5	0.0069 ± 0.0405	0.3922 ± 0.7051
1,1,1,1	0.0118 ± 0.0856	2.5530 ± 4.0324
-.5,1,-.5,-.5	0.0095 ± 0.0478	0.0556 ± 0.1041
-1,1,0,1	0.0014 ± 0.0264	0.0652 ± 0.2470

Table 1: **MSFA has a smaller reward prediction error for test tasks**. We present the reward prediction error for MSFA and USFA-Learned-ϕ. We present results using 40 episodes for each test task.

B Combining knowledge of heterogeneous tasks

Aside from navigation tasks, researchers and practitioners will be interested in generally combining solutions to different tasks. **R5**: Can MSFA enable combining solutions to heterogeneous tasks?

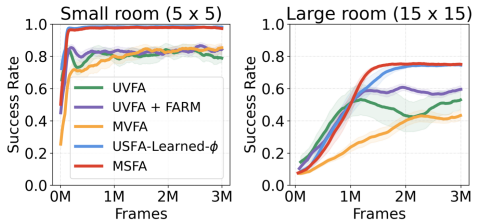

Figure 9: We study whether MSFA can combine three separate settings (1) limited visibility (2) the presence of monsters (3) the presence of teleportation traps (see text for more). All successor feature based methods best enable combining training tasks for generalization in this setting. (10 run)

Setup. We leverage the "Minihack" environment (Samvelyan et al., 2021). Here, an agent is spawned at the top of a map with obstacles. It must navigate to the a ladder at the bottom of the map. The agent experiences partial **observations** of the environment. **Actions**: At each time-step the agent can select from 8 compass directions. **Training tasks**: The agent experiences three training tasks (1) avoiding monsters which kill the agent, (2) avoiding traps which teleport the agent away from the goal, and (3) experiencing a small local view of the environment with everything else black. **Generalization** requires that the agent generalize to (a) new configurations of objects and (b) to the *combination* of all training conditions.

R5: MSFA enables combining solutions to heterogeneous tasks. Figure 9 shows that SF-based methods best combine solutions to different tasks. In a small room, both SF-based methods get 100% success rate. In a larger room, no method generalizes perfectly: SF-based methods do best at 75%, followed by UVFA-FARM and UVFA with 60%.

C Derivation of Modular Successor Features

$$Q^{\pi}(s,a,w) = \mathbb{E}_{\pi}\left[\sum_{j=0}^{\infty}\gamma^{j}R_t|S_t=s,A_t=a\right] \tag{9}$$

$$= \mathbb{E}_{\pi}\left[\sum_{j=0}^{\infty}\gamma^{j}\left(\sum_{k=1}^{n}{\phi_{t+j}^{(k)}}^{\top}w^{(k)}\right)|S_t=s,A_t=a\right] \tag{10}$$

$$= \sum_{k=1}^{n}\left(\mathbb{E}_{\pi}\left[\sum_{j=0}^{\infty}\gamma^{j}{\phi_{t+j}^{(k)}}^{\top}w^{(k)}|S_t=s,A_t=a\right]\right) \tag{11}$$

$$= \sum_{k=1}^{n}\left(\mathbb{E}_{\pi}\left[\sum_{j=0}^{\infty}\gamma^{j}\phi_{t+j}^{(k)}|S_t=s,A_t=a\right]^{\top}w^{(k)}\right) \tag{12}$$

$$= \sum_{k=1}^{n}\psi^{\pi,k}(s,a)^{\top}w^{(k)} \tag{13}$$

$$= \sum_{k=1}^{n}Q^{\pi,k}(s,a,w^{(k)}) \tag{14}$$

D Inter-module attention and gating

At each time-step t, each module updates with both observation features $z_t = f_z^{\theta}(x_t)$ and with information from the previous module-states $\{s_{t-1}^{(k)}\}$. MSFA shares information between

modules with a "query-key" system. Each module has a "query" representation of its module-state that it uses to select from "key" representations of the other module-states. MSFA scores how well a query matches a key by computing the dot-product between the two representations. A module then updates with the module-state corresponding to the key with the highest dot-product. To enable flexible updating, each module uses a gating mechanism to decide the degree to which information should be incorporated into an update.

More technically, the query vector is computed using the previous-module state and action: $q_t^{(k)} = W^{\texttt{query}}[s_{t-1}^{(k)}, a_{t-1}] \in \mathbb{R}^{d_q}$. Keys and values are computed as $K_t = W^{\texttt{key}}[s_{t-1}^1; \ldots; s_{t-1}^n; 0] \in \mathbb{R}^{n+1 \times d_q}$ and $V_t = W^{\texttt{value}}[s_{t-1}^1; \ldots; s_{t-1}^n; 0] \in \mathbb{R}^{n+1 \times d_q}$, respectively. Note that we add a zero-vector key and value to allow a query to select "no information". Each module selects "values" $v_t^{(k)}$ to update using dot-product attention (Vaswani et al., 2017):

$$v_t^{(k)} = \text{softmax}\left(\frac{q_t^{(k)} K_t^\top}{d_q}\right) V_t \tag{15}$$

Gating mechanism has been shown important for leveraging transformer-style attention when updating state in RL agents (Parisotto et al., 2020). Thus, each module uses a sigtanh gate when updating:

$$u_t^{(k)} = q_t^{(k)} + \tanh(W^{g_1} v_t^{(k)}) \odot \sigma(W^{g_2} v_t^{(k)} - b_g). \tag{16}$$

MSFA then updates its module-stateas follows:

$$s_t^{(k)} = s_{\theta_k}(u_t^{(k)}, z_t) \tag{17}$$

E Learning modular functions

Our goal is to learn functions for producing cumulants ϕ and SFs ψ that have consider only state information from their own modules. To be more precise, we learn modules which collectively learn a set of n module-state representations: $\{s^i\}_i^{n=1}$. As an example, consider learning a linear function/transformation A for producing cumulants. A typical "monolithic function" would be one like the following:

$$\begin{bmatrix} \phi^1 \\ \vdots \\ \phi^n \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & A_{22} & \vdots \\ A_{n1} & \cdots & A_{nm} \end{bmatrix} \begin{bmatrix} s^1 \\ \vdots \\ s^n \end{bmatrix}.$$

By "entangled," we mean that a cumulant ϕ^i is (potentially) a function of all module-state information $\phi^i = \sum_j A_{ij} s^j$. By a "modular function", we simply meant that we were learning a function like the following

$$\begin{bmatrix} \phi^1 \\ \vdots \\ \phi^n \end{bmatrix} = \begin{bmatrix} A_1 & 0 & 0 \\ 0 & A_2 & 0 \\ 0 & 0 & A_n \end{bmatrix} \begin{bmatrix} s^1 \\ \vdots \\ s^n \end{bmatrix}.$$

In our setting, we had $A_1 = \ldots = A_n$ be a shared MLP and each s^i was produced by a module with its own parameters. Our experiments demonstrate leveraging modules to learn $\phi^{(i)}$ and $\psi^{(i)}$ is an effective way to learn $\{\phi^i\}_i$ that promote zero-shot composition of task knowledge with the SF&GPI framework.

F Additional analysis

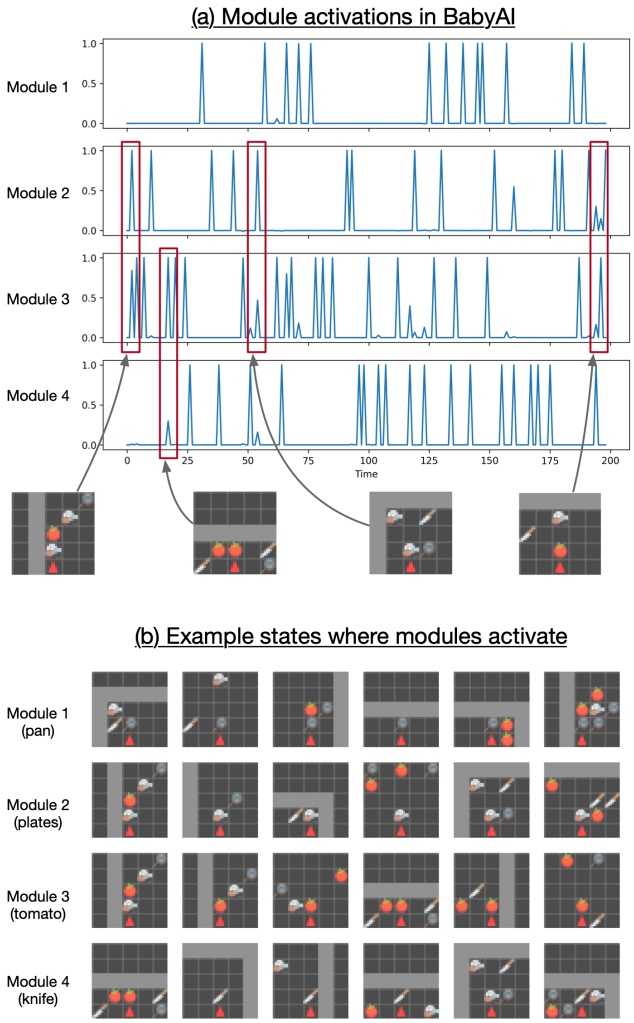

Figure 10: We present a visualization of module activity for task generalization task $w = [1, 1, 1, 1]$ from §5.1. On the top, (a) we show the activity of each module across time. On the bottom, (b) we show example transitions where modules activate to visualize what they respond to. We see that individual modules are able to effectively respond to different object categories, despite a strong data imbalance involved in learning to represent each object category (see §). We find that modules don't perfectly specialize, and multiple modules will sometimes activate at the same state.

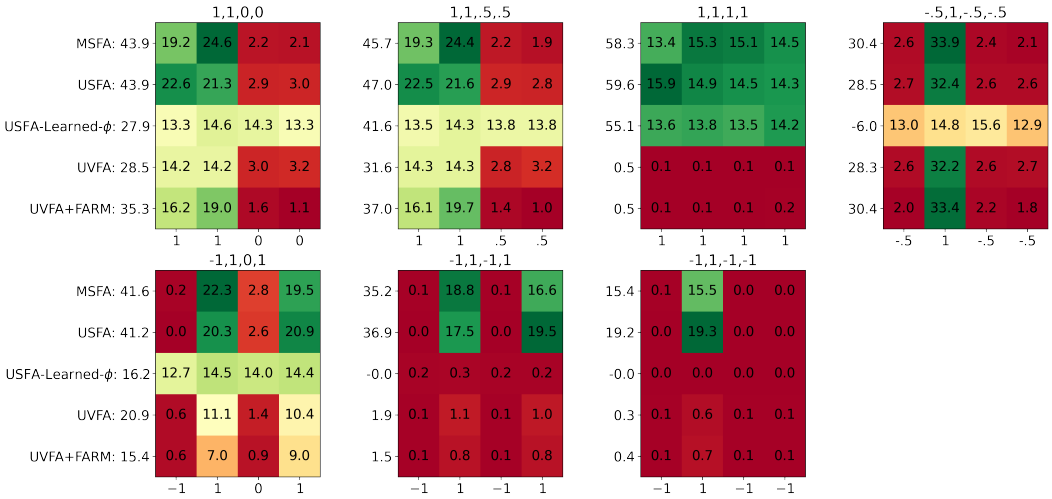

Figure 11: **MSFA best matches the behavior of USFA across generalization tasks**. We present additional analysis for the experiments in §5.1. Each heatmap displays how often object categories were picked up by each method for a given generalization task. Rows correspond to different methods. We show the final episode return for each method along the y-axis. Columns describe the transition-reward when an object category is picked up. USFA has access to hand-designed cumulants that described whether an object category was picked up. Despite learning cumulants, MSFA best matches USFA's object collection dynamics. USFA-Learned-ϕ equally collects all objects regardless of task, except for when $r \leq -1$ when an object is picked up. In this setting, no method except USFA or MSFA collect objects. As a reminder, negative rewards are never experienced during training and these tasks are the furthest away from training tasks in task-space. UVFA-based methods tend to collect rewarding objects but not as effectively as MSFA or USFA.

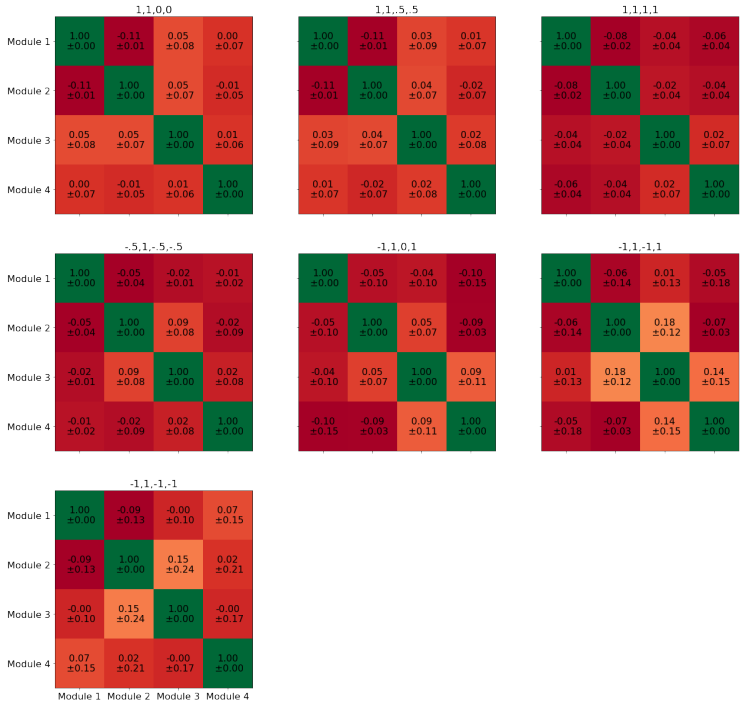

Figure 12: We present the cross-correlation between pairs of modules. This confirms that modules tend to activate at different time-points on a statistical level.

G Full results

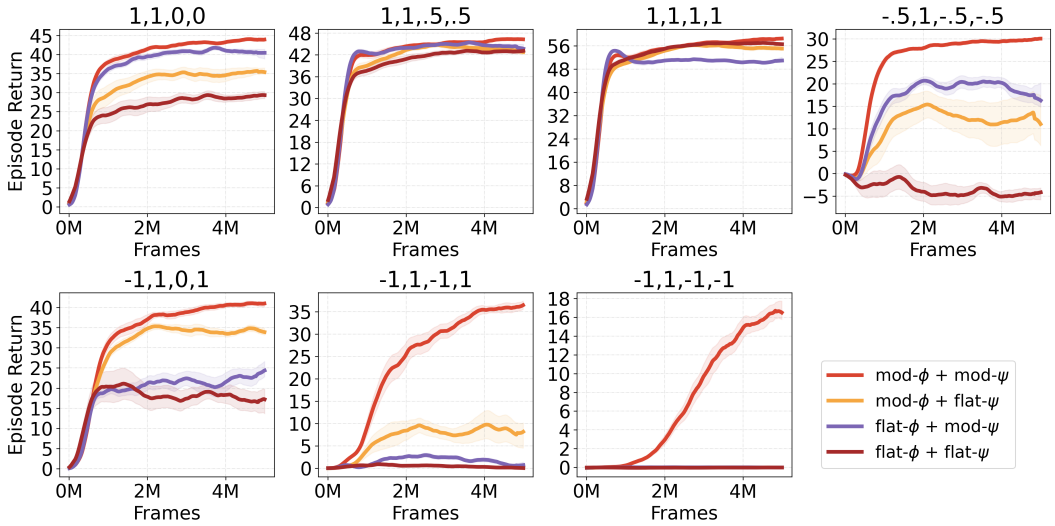

Figure 13: All results for ablating disentanglement of ϕ and ψ. We consistently find that disentangled functions for ϕ and ψ have the best generalization performance.

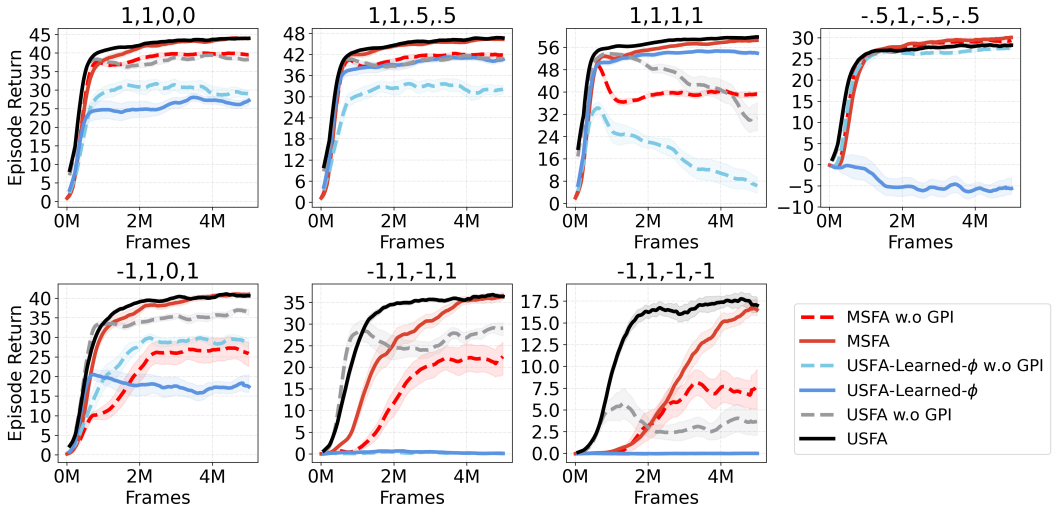

Figure 14: All results for ablating GPI. First, we compare MSFA to USFA (which has hand-designed cumulants). We can see that the two perform comparably across the board. Sometimes (e.g. $[1, 1, 1, 1]$ and $[-1, 1, -1, -1]$), MSFA outperforms USFA. Next we compare MSFA to USFA-Learned-ϕ. We see that MSFA performs as well as USFA-Learned-ϕ or better in all settings except $[-1, 1, 0, 1]$. Interestingly, MSFA without GPI outperforms USFA-Learned-ϕ with GPI in some generalization settings (e.g. $[1, 1, 0, 0]$ and $[-1, 1, -1, 1]$).

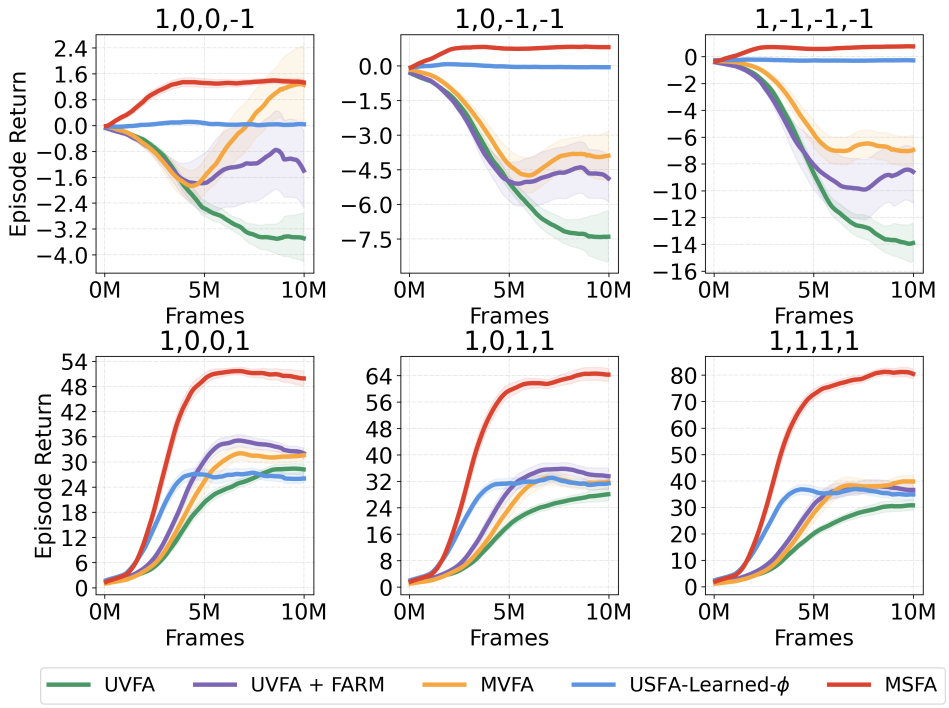

Figure 15: All results for the Fruitbot environment. When objects must be avoided, we see that no method does well, though MSFA generalizes best and can pick up a couple of objects. When combinations of objects must be collected, MSFA outperforms baselines by a large margin.

H Hyperparameters

We shown hyperparameters in Tables 2 and 3. Tables 2 describes hyperparameters that were shared across algorithms and Tables 3 describes hyperparameters that were different across algorithms. We developed the algorithms using the Hoffman et al. (2020) reinforcement learning codebase and unless stated otherwise using default "R2D2" (Kapturowski et al., 2018) config values in the codebase. We describe our development and hyperparameter search in more detail below.

H.1 Development and hyperparameter search

The first thing we did was replicate the generalization gap between USFA and UVFA that Borsa et al. (2019) produced in their paper using the BabyAI gridworld (Chevalier-Boisvert et al., 2018). As a result, most hyperparameters were tuned for either USFA or UVFA. We used the Atari Convolutional network used by "Deep Q-Networks" (DQN) in Mnih et al. (2015) as our visual encoder and an LSTM (Hochreiter & Schmidhuber, 1997) as our state representation function.

Once we were able to replicate their generalization performance, we implemented USFA-Learned-ϕ and MSFA. In this setting, we are learning Q-values, successor features, and predicting rewards from cumulants. This leads to the following coefficients for losses: α_Q, α_ψ, and α_ϕ respectivey. We fixed $\alpha_\psi = 1$ and searched $\alpha_\phi \in [.01, .1, 1, 10, 100]$ and $\alpha_Q \in [.01, .05, .1, .5, 1.0]$ for both USFA-Learned-ϕ and MSFA. For MSFA and UVFA-FARM, we chose the number of modules to be the number of training tasks. We set each module size so that the number of parameters by all algorithms was approximately the same (with USFA as a reference). We found that FARM hyperparameter (Carvalho et al., 2021a) worked well for both UVFA-FARM and for MSFA (FARM modules are the basis of MSFA state-modules). We highlight that properly masking all losses was absolutely critical to good generalization performance. Without proper masking, out-of-episode data is used for value estimation of in-episode data, which can lead to inaccurate value estimates. We suspect that this hampers GPI.

When doing experiments for Fruitbot, we took all implementation details from (Cobbe et al., 2020). We first replicated the DQN performance from Cobbe et al. (2020) with UVFA. Some important changes that we made included (1) adding prioritized experience replay (2) leveraging the Impala ResNet vision torso (Espeholt et al., 2018) (3) lowering the memory size to match the paper (4) using the `SIGNED_HYPERBOLIC_PAIR` value transformation for UVFA-based algorithms (5) increasing the capacity of the Q-value estimation MLPs. We applied these settings to other baselines and redid our search over α_ϕ and α_Q for both MSFA and USFA-Learned-ϕ. For each method, we changed the size of the networks so they were approximately the same (with UVFA as a reference). Since MSFA uses disentangled functions for ϕ and ψ, it has fewer parameters.

When doing experiments in Minihack, we found that the Fruitbot changes were important for good performance in this domain. All hyperparameters led to strong training performance but poor generalization for UVFA. We found that increasing the batch size was important for improving generalization of USFA-Learned-ϕ and MSFA but was detrimental to UVFA-based methods.

I Environments

We presented most environment information in the main text. Here, we specify which levels we used from the corresponding environments or how the changed environments for our experiments.

I.1 Procgen

We used the Fruitbot environment in Procgen. We made the following changes. We set the maximum number of levels that an agent could complete within a lifetime to 4. We divided the

Table 2: Hyperparameters shared across all algorithms.

Algorithm			
Loss Hyperparameters	BabyAI	Fruibot	Minihack
discount	0.99	0.99	0.99
burn_in_length	0	0	0
trace_length	40	40	40
importance_sampling_exponent	0	0.6	0.6
max replay size	100,000	70,000	70,000
max gradient norm	80	80	80
Shared Network Components			
Vision torso	DQN ConvNet	Impala ResNet	Impala ResNet

Table 3: Hyperparameters for individual algorithms. Note: T_1 = `IDENTITY_PAIR` and T_2 = `SIGNED_HYPERBOLIC_PAIR`.

	BabyAI	Fruibot	Minihack
MSFA			
Parameters (millions)	2.1M	1.66M	1.7M
α_ϕ	1	1	1
α_ψ	1	1	1
α_Q	0.5	0.5	0.5
module_size	150	60	80
nmodules	4	4	3
attention heads	2	2	1
batch size	32	32	64
ϕ MLP hidden sizes	[256]	[256]	[256]
ψ MLP hidden sizes	[128]	[512, 512]	[512, 512]
projection dim	16	16	16
USFA			
Parameters (millions)	2.35M	1.98M	1.95M
lstm size	512	300	300
α_ϕ	1	1	1
α_ψ	1	1	1
α_Q	0.5	0.5	0.5
batch size	32	32	128
ϕ MLP hidden sizes	[256]	[256]	[256]
ψ MLP hidden sizes	[128]	[512, 512]	[512, 512]
UVFA			
Parameters (millions)	2.09M	1.96M	1.95M
lstm size	512	256	256
rlax.TxPair	T_1	T_2	T_2
batch size	32	32	32
Q MLP hidden sizes	[128]	[512, 512]	[512, 512]
UVFA-FARM			
Parameters (millions)	2.17M	2.15M	2.06M
module_size	150	64	80
nmodules	4	4	3
attention heads	2	2	1
rlax.TxPair	T_1	T_2	T_2
batch size	32	32	32
projection dim	16	16	16
Q MLP hidden sizes	[128]	[512, 512]	[512, 512]

12 object types $\{O_1 \dots O_{12}\}$ into 4 categories: $C_1 = \{O_1, O_2, O_3\}, \dots, C_4 = \{O_{10}, O_{11}, O_{12}\}$, where each category provided postive reward for picking up any of its 3 category types.

I.2 MiniHack

We did not make any changes to the environment. Train tasks were: "MiniHack-Room-Monster-Y-v0", "MiniHack-Room-Trap-Y-v0", and "MiniHack-Room-Dark-Y-v0". Test tasks were "MiniHack-Room-Ultimate-Y-v0". We used $Y = 5 \times 5$ for the "small room" experiments and $Y = 15 \times 15$ for the "large room" experiments.