A Reproducibility

A.1 Environment description

We run experiments on two benchmarks as described in Table 1. All these benchmarks are implemented in MuJoCo [Todorov et al., 2012]. Huang et al. [2020] created `Walker++`, `Humanoid++`, `Hopper++` and `Walker-Humanoid-Hopper++`, by removing some limbs and joints from the original intact morphology of Gym MuJoCo robots [Todorov et al., 2012, Brockman et al., 2016]. In this benchmark, every robot has the ability to hop, walk, or run, *i.e.*, only the robots that can move forward are left. Following Huang et al. [2020], Kurin et al. [2020], Hong et al. [2021], we use this benchmark for the evaluation of multi-task learning (Sec. 4.2) and zero-shot generalization ability (Sec. 4.3). As for the single-task learning benchmark, we sampled some morphologies from UNIMALS [Gupta et al., 2021b,a]. Gupta et al. [2021b] created UNIMALS by evolution on robots' morphologies, kinematics, and dynamics. At the end of the evolution, they had 100 task optimized robots. We choose a subset of these robots that are optimized for locomotion in flat terrain environment for single-task evaluation in Sec. 4.5. We ensure that the selected robots have different morphologies from each other. For multi-task evaluation, we use the whole 100 robots for training in Appendix B.1. And for zero-shot evaluation in robots' dynamic and kinematic properties in Appendix B.1, we use the test set created by Gupta et al. [2021a], which contains 2400 robots.

The reward of these tasks is given by the speed of the robot moving forward, and is penalized by the action norm. An episode terminates when the robot's height is low or the episode length exceeds 1000 time steps. For all the environments, we can extract the morphology of the robot as a graph from the corresponding MuJoCo XML file. Then the morphological information is concatenated in observations in baselines or is used to generate embeddings for action transformation in SOLAR. The visualization of these robots can be found in synergy clustering results: `Walker++` as in Figure 11, `Humanoid++` as in Figure 5, `Hopper++` as in Figure 10 and UNIMALS as in Figure 12. The number of actuators we evaluated in this work varies from three to nine.

Table 1: Full list of environments.

Environment	Train set	Test set
`Walker++`	walker_2_main walker_4_main walker_5_main walker_7_main	
`Humanoid++`	humanoid_2d_7_left_arm humanoid_2d_lower_arms humanoid_2d_7_right_leg humanoid_2d_8_left_knee humanoid_2d_8_right_knee humanoid_2d_9_full	humanoid_2d_7_left_leg humanoid_2d_7_right_arm
`Hopper++`	hopper_3 hopper_4 hopper_5	
`Walker-Humanoid-Hopper++`. Union of `Walker++`, `Humanoid++` and `Hopper++`		
UNIMALS	unimalA, unimalB, unimalC	

A.2 Implementation details

We implement SOLAR based on AMORPHEUS [Kurin et al., 2020], which is built on Official PyTorch Tutorial [Seq, 2020]. AMORPHEUS also shares the codebase with SMP [Huang et al., 2020]. Table 2 provides the hyperparameters needed to replicate our experiments.

As in Figure 1, there are three main components in our implementation, *i.e.*, *Intra-Synergy Self-Attention*, *Inter-Synergy Self-Attention* and *Action Transformation*. To achieve *Intra-Synergy Self-Attention* efficiently, we pass a synergy-based mask into PyTorch's *TranformerEncoderLayer*

Table 2: Hyperparameters of our SOLAR.

Hyperparameter	Value
Learning rate	0.0001
Gradient clipping	0.1
Normalization	LayerNorm
Total attention layers	3
Intra-synergy attention layers	1
Inter-synergy attention layers	2
Attention heads	2
Attention hidden size	256
Encoder output size	128
Mini-batch size	100
Replay buffer size	10M
Attention embedding size	128

through *src_mask* function argument. The synergy-based mask is discussed in detail in Sec. 3.2. *Inter-Synergy Self-Attention* is implemented by the normal *TranformerEncoderLayer* without masks.

As for *Action Transformation*, the main challenge is how to obtain a transformation matrix $\mathbf{T} \in \mathbb{R}^{K_n \times L_n}$, where K_n and L_n are the number of actuators and the number of synergies of robot $n \in \{1, 2, \cdots, N\}$. L_n may change during the learning process. Intuitively, the matrix $\mathbf{T}$ is dependent on the robot's morphology. Here we use a traversal-based positional embedding [Hong et al., 2021] technique, and these embeddings together represent the robot's morphological information. To obtain the traversal-based positional embedding, we first apply left-child-right-sibling representation to represent a general tree as a binary tree. Then, we traverse the binary tree in pre-order, in-order and post-order, which forms a triple for each actuator consisting of its orders. And these triples together are sufficient to reconstruct the original tree, which indicates that each triple contains the structural information of the corresponding actuator. The triple of each joint serves as indices into a embedding pool $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_p\}$ where $p = \max_n K_n$ is the maximum number of actuators in a robot, and each $\boldsymbol{r}_i \in \mathbb{R}^s$ is a learnable vector. The selected embeddings are processed by a network, and we concatenate the outputs to obtain a representation of the corresponding actuator. Using dot product between representations of two actuators, we get a matrix $\mathbf{H} \in \mathbb{R}^{K_n \times K_n}$ where $\mathbf{H}_{i,j}$ indicates the relation between actuator i and actuator j. Finally, we average the columns corresponding to actuators in the same synergy cluster and obtain the transformation matrix $\mathbf{T} \in \mathbb{R}^{K_n \times L_n}$. The embedding pool and network are updated by backpropagating the RL loss.

Experiments are carried out on NVIDIA GTX 2080 Ti GPUs. For `Humanoid++`, our method requires approximately 10G of RAM and 5G of video memory, and takes about 23 hours to finish 2M timesteps of training.

The code for our method is included in the supplementary materials with sufficient setup and running instructions. The code follows the MIT license.

A.3 Baselines and ablations

We compare our method against various baselines and ablations. Here we explain the implementations of these baselines and ablations.

AMORPHEUS. We use the original implementation of AMORPHEUS released by Kurin et al. [2020]. For fair comparison, SOLAR uses the same value for those hyperparameters that are shared with AMORPHEUS (Table 2).

SMP. We use the implementation of SMP in the AMORPHEUS codebase, which is the same as the original implementation of SMP provided by Huang et al. [2020].

Monolithic. Following the setup by Huang et al. [2020], we choose TD3 as the standard monolithic RL baseline. The actor and critics of TD3 are implemented by fully-connected neural networks, which take the concatenation of observations of all actuators as input. Since number of actuators

varies in different robots, the dimension of observations is incompatible. To overcome this issue, we zero-pad the observations and actions to the maximum dimension across all robots.

AMORPHEUS *w/ synergy mask*. We passed a synergy-based mask to the first layer of self-attention in AMORPHEUS, and the mask is learned in the same way as in our method. The other two layers of AMORPHEUS are not modified.

SOLAR *w/o preference*. We simply set the preference vector of the affinity propagation clustering algorithm to None, and the algorithm will use the median of the input affinity matrix as a default preference vector.

B Additional experiments and discussions

B.1 UNIMALS Locomotion task and Manipulation task

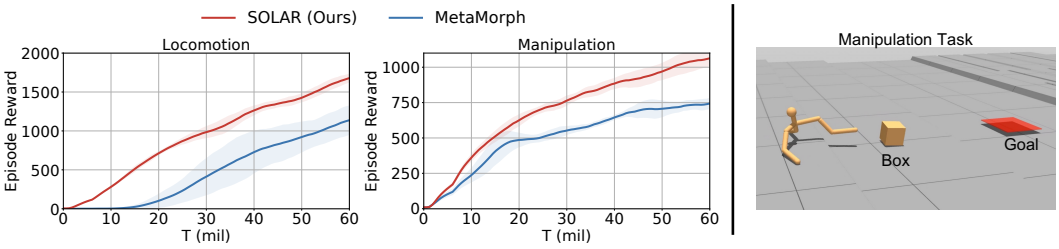

Figure 8: Left: Multi-Task performance in UNIMALS of our method SOLAR compared to MetaMorph. Right: Manipulation task.

To further analyze the scalability of our SOLAR when the number of robots and the number of actuators are large, we use the original UNIMALS [Gupta et al., 2021b] Locomotion tasks in flat terrain and Manipulation tasks in variable terrain. In Locomotion tasks, the robot needs to go forward as fast as possible. In Manipulation tasks, the robot needs to first reach a box, and then push this box to a randomly generated goal. The UNIMALS has 100 robots with different morphologies, kinematics, and dynamics for multi-task training. We compared our method SOLAR against MetaMorph [Gupta et al., 2021a]. In this subsection, SOLAR is built upon MetaMorph. Like MetaMorph, SOLAR incorporates structural information into AMORPHEUS and uses a dynamic replay buffer balancing technique to deal with the large number of robots. SOLAR can be simply combined with MetaMorph by modifying its self-attention structure. The training results are shown in Figure 8. SOLAR outperforms MetaMorph by a large margin, which suggests that SOLAR is also effective with numerous different robots.

Table 3: Zero-shot generalization performance of SOLAR compared against MetaMorph in UNIMALS Locomotion task.

	Variants	SOLAR	MetaMorph
Dynamics	Armature	**1253.39** $\pm$ 28.25	843.31 $\pm$ 14.80
	Density	**1654.43** $\pm$ 24.55	1089.64 $\pm$ 22.64
	Damping	**1663.64** $\pm$ 26.21	1113.04 $\pm$ 10.18
	Gear	**1480.45** $\pm$ 30.49	987.29 $\pm$ 8.05
Kinematics	Module param.	**1084.41** $\pm$ 13.93	700.43 $\pm$ 17.02
	Joint angle	**477.50** $\pm$ 6.83	274.37 $\pm$ 6.55

In Sec. 4.3, we test the zero-shot performance of SOLAR on robots with different morphologies. However, in reality, there are cases where the morphologies are not changed, but the dynamics (armature, density, damping, and motor gear) and kinematics (module shape parameters and joint angles) of robots are new in unseen tasks. Thus we also benchmark the zero-shot performance of SOLAR in these cases. Gupta et al. [2021a] created a test set to test zero-shot performance based on the 100 training robots of UNIMALS. For each robot in the training set, they create 4 different variants for each property (regarding the dynamics and kinematics), leading to a total number of

Table 4: Zero-shot generalization performance of SOLAR compared against MetaMorph in UNIMALS Manipulation task.

	Variants	SOLAR	MetaMorph
Dynamics	Armature	**792.69** $\pm$ 32.35	685.36 $\pm$ 9.03
	Density	**943.07** $\pm$ 57.46	823.20 $\pm$ 18.95
	Damping	**984.19** $\pm$ 24.14	860.70 $\pm$ 11.24
	Gear	**862.04** $\pm$ 36.90	784.54 $\pm$ 12.01
Kinematics	Module param.	**663.18** $\pm$ 12.47	611.43 $\pm$ 5.80
	Joint angle	314.16 $\pm$ 4.95	**455.49** $\pm$ 11.28

Table 5: Dynamics and kinematics variations to generate the test robot set, reproduced from Gupta et al. [2021a].

Kinematics	
Variation type	**Value**
Limb radius	[0.03, 0.05]
Limb height	[0.15, 0.45]
Joint angles	[(30, 0), (0, 30), (30, 30), (45, 45), (45, 0), (0, 45), (60, 0), (0, 60), (60, 60), (90, 0), (0, 90), (60, 30)(30, 60)]
Dynamics	
Armature	[0.1, 2]
Density	[0.8, 1.2] $\times$ limb density
Damping	[0.01, 5.0]
Gear	[0.8, 1.2] $\times$ motor gear

$100 \times 6 \times 4 = 2400$ test variants. Here we reproduce from their paper the sampling ranges of test variants in Table 5 . Please refer to Gupta et al. [2021a] for more details. Due to the limitation of computing resources, we reduce the batch size of SOLAR and MetaMorph from 5120 to 128, and other hyperparameters follow Gupta et al. [2021a].

To evaluate the zero-shot generalization performance, we load our models trained on multi-task UNIMALS and show the averaged performance (and variance) over 4 random seeds in Table 3 and Table 4 for Locomotion task and Manipulation task, respectively. We find that SOLAR exhibits strong generalization performance compared to the baseline, which indicates that the synergy clusters and action transformations learned on the training set are robust to dynamic and kinematic variations.

B.2 Compared with other baselines in single-task

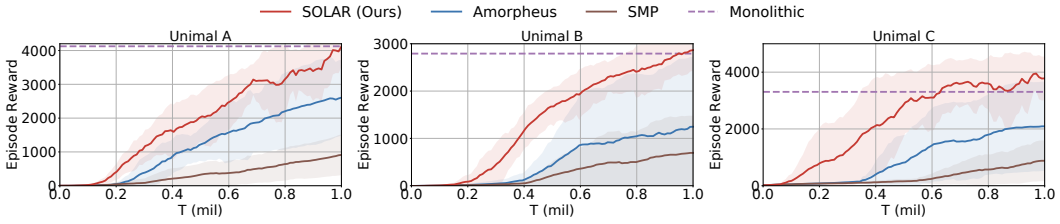

Figure 9: Multi-Task performance of our method SOLAR compared to AMORPHEUS, SMP and the final performance of Monolithic.

In Sec. 4.5, we compare SOLAR with transformer-based baseline AMORPHEUS, here we also compare our SOLAR with the GNN-style baseline SMP in single-task evaluations in Figure 9 as additional results.

B.3 Synergy clustering results of more morphologies

In Figure 5, we visualize the synergy clustering results of `Humanoid++` to analyze the effect of synergy. In this section, we provide visualizations of synergy clustering results for `Hopper++` (Figure 10), `Walker++` (Figure 11), and UNIMALS (Figure 12) as additional results.

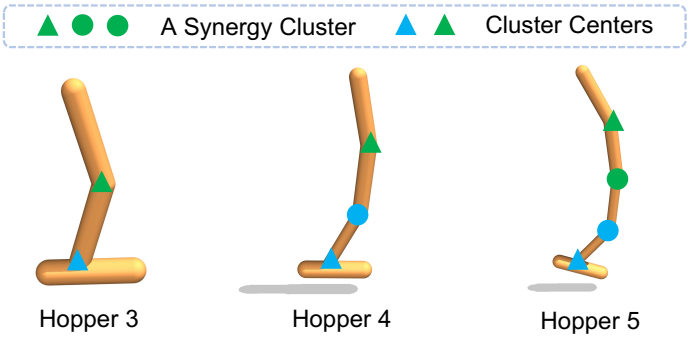

Figure 10: Synergy clustering results of SOLAR in `Hopper++`. Different colors represent different synergy clusters, and joints marked with the same color are in the same cluster. Joints marked with triangles are the centers of their corresponding clusters.

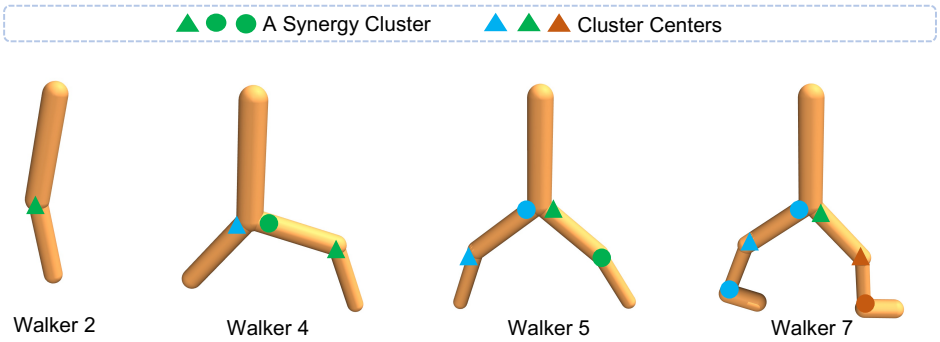

Figure 11: Synergy clustering results of SOLAR in `Walker++`.

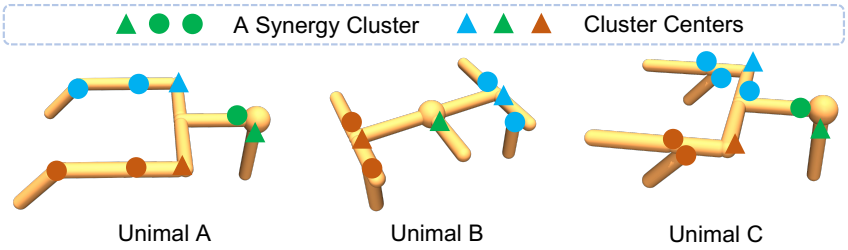

Figure 12: Synergy clustering results of SOLAR in UNIMALS.

These additional results further consolidate our analysis in Sec. 4.4: (1) close joints are more likely to be in the same synergy cluster, and (2) joints near the torso may be more influential than those who are far from the torso, and are thus selected as the cluster centers.

B.4 Connection to conventional notion of muscle synergy

Mathematically, conventional muscle synergy studies typically use non-negative matrix factorization (Rabbi et al, 2020) and aim to solve the following optimization problem:

$$\min_{H \in \mathbb{R}_+^{N \times M}, A \in \mathbb{R}_+^{M \times T}} \|U - HA\|_F,$$

where $U \in \mathbb{R}^{N \times T}$ is a given matrix of observed electrical control signals. The element at ith row and tth column of U is the control signal to muscle i at timestep t. In this optimization problem, the num-

ber of synergies, M (where $M < N$), is typically pre-defined or chosen according to a pre-defined reconstruction error threshold. By solving this problem, one can discover the synergy structure by observing matrix H. Conventional muscle synergy studies that use other factorization methods (PCA, ICA, and FA) share the similar optimization problem with different matrix constraints.

We study a similar optimization problem but with additional constraints:

$$\max_{U,H,A} \sum_t \gamma^t R(s_t, U_t) - \|U - HA\|_F.$$

Here s_t is the environment state at timestep t, and U_t is the column t of matrix U, i.e., actions at timestep t. And $R(s_t, U_t)$ is the reward of choosing action U_t at state s_t.

The differences are:

1. We additionally maximize the expected return of muscle actions.
2. In our formulation, the number of synergies is not pre-defined but is learned in an unsupervised manner.
3. The matrix U is also not given, but is generate by an attention-based policy. This policy are optimized to maximize return as well as to minimize the decomposition loss (term 2).

In summary, we share a common optimization object with conventional muscle synergy studies, which is a decomposition loss. However, we need to additionally learn the number of synergies and control signals which are inputs in the conventional studies. Structurally, our framework gives an attention function class that covers a synergy decomposition solution which can minimize the decomposition loss while enable an efficient control policy.

B.5 Limitations and future works

When generalizing to unseen robots with larger numbers of actuators than training robots, the embeddings of testing robots are not learned, which will hamper the zero-shot performance. One possible future direction will involve designing a more scalable actuator embedding method. Moreover, SOLAR is more suitable for robots with large number of actuators. Our approach is able to reduce a great degree of control complexity for these robots. But for robots with few actuators, SOLAR may only have a little positive impact and the learning of synergy-aware policy may even damage the performance.