

Figure 8: These figures show tasks use in the **MetaWorld** environment.

A Experimental Details

Here we provide additional information on the task sequences and reward functions for each environment.

Ant Goal.

1. Goal description: With the agent starting at the origin ($x = 0, y = 0$), goal locations are sampled 3 units away from the origin uniformly on a circle.
2. Non-stationary task sequence: We fixed twenty tasks, $[-36^\circ, -126^\circ, 0^\circ, 162^\circ, -108^\circ, -144^\circ, 36^\circ, 90^\circ, -90^\circ, 18^\circ, -180^\circ, 72^\circ, 108^\circ, 144^\circ, -72^\circ, 54^\circ, -18^\circ, -162^\circ, 126^\circ, -54^\circ]$, where each number represents the rotation of the goal’s position, where for 0 the goal position of ($x = 3, y = 0$) and 90 represents goal position of ($x = 0, y = 3$). The stationary task distribution randomly shuffles this sequence.
3. Maximum Trajectory length T^{max} : We set $T^{max} = 200$, same as GMPS [30].
4. Reward function: We modified the reward function in PEARL [37], reducing the control cost portion of the reward by an order of magnitude. This is accomplished by multiplying the control value by 0.1.

Ant Direction:

1. Goal description: With the agent starting at the origin, goal directions, as vectors (x, y), are sampled uniformly.
2. Non-stationary task sequence: We fixed twenty tasks, $[-36^\circ, -126^\circ, 0^\circ, 162^\circ, -108^\circ, -144^\circ, 36^\circ, 90^\circ, -90^\circ, 18^\circ, -180^\circ, 72^\circ, 108^\circ, 144^\circ, -72^\circ, 54^\circ, -18^\circ, -162^\circ, 126^\circ, -54^\circ]$, where each number represents the degree of the goal’s angle, e.g. 0 represents a goal that is aligned with the positive x-axis and 90 represents positive y-direction. For stationary distribution experiments the sequence is shuffled randomly.
3. Maximum Trajectory length T^{max} : We set $T^{max} = 200$, same as GMPS
4. Reward function: We modified the reward functions from the version of the environment used in PEARL by adding a scaling factor of 0.1 to the control cost.

Half Cheetah:

1. Goal description: The target velocities can be in the range $[-3, 3]$.
2. Non-stationary task sequence: We fixed twenty tasks described by there goal velocity along the x-axis $[0.8, -1.20, 1.00, -0.80, 0.40, 1.20, 1.60, -1.60, 2.00, -2.00, 1.40, -1.40, 2.80, -2.80, 2.99, -2.99, 0.50, -0.50, -0.40, -1.00]$. For the stationary task distribution the above sequence is randomly shuffled.
3. Maximum Trajectory length T^{max} : We set $T^{max} = 200$, same as GMPS [30].
4. Reward function: There were no changes to the reward function from the environment used in PEARL [37].

628 MetaWorld:

- 629 1. Goal description: We chose three environments from Metaworld: Push-Wall-V2, Button-
630 Press-Wall-V2, and Hammer V-2. These environments were chosen based on their dissimilar-
631 ity and difficulty measured by the level of *success* regular PPO could achieve on these tasks.
632 For each task, we specify both the environment (Push-Wall-V2 (P), Button-Press-Wall-V2
633 (B), or Hammer V-2 (H)) and the goal index of that environment. The goal index controls
634 additional variation in the tasks. For example, in the Hammer v-2 environment, shown
635 in Figure 8 different goal tasks change the location of the hammer at the start of the episode.
- 636 2. Non-stationary task sequence: We fixed twenty-one tasks, [H-0, H-1, B-2, B-3, P-4, P-5,
637 H-6, H-7, B-8, B-9, P-10, P-11, H-12, H-13, B-14, B-15, P-16, P-17, H-18, B-19, P-20],
638 where the letter on the left of '-' represents the environment ('H' for Hammer-V2, 'B'
639 for Button-Press-Wall-V2, and 'P' for Push-wall-V2), while the number to the right of '-'
640 specifies the goal index used in the environment. For the stationary task distribution, the
641 above sequence is randomly shuffled depending on the random seed.
- 642 3. Maximum Trajectory length T^{max} : We set $T^{max} = 150$, same as the original Metaworld
643 codebase [60].
- 644 4. Reward function: We used the same functions for the three environments as the original
645 Metaworld codebase.

646 **Success Thresholds** For the experimental evaluation, we are interested in measuring how quickly
647 an agent can solve tasks. Previous meta-learning environments [60] have used a measure of *success*
648 to determine the agent's progress on its tasks. These success measures are more robust to tasks that
649 can have largely different reward scales. Each environment includes a success indicator that is 1
650 when the agent is within ϵ distance of the desired task goal and 0 otherwise. In this training version,
651 the agent moves on to the next task when it has been told it has succeeded on the current one. This
652 setting makes particular sense in the continual multi-task setting, where the goal is to do well on all
653 tasks with minimal interaction.

- 654 1. Ant Goal: The agent is *successful* if the agent's position (a_x, a_y) and the goal's position
655 (g_x, g_y) satisfy $(|a_x - g_x| + |a_y - g_y| < 0.5)$. We calculate the number of successful
656 samples from a batch of trajectories and average the success across all samples. We found
657 that agents trained through PPO training could reach a success rate of slightly over 0.3. We
658 therefore set the success rate *threshold* to 0.3.
- 659 2. Ant Direction: The agent is successful if its unit direction vector v_a and the unit goal
660 direction vector v_g satisfy $\|v_a - v_g\| < 0.4$ while at the same time, the agent's speed must
661 be greater than 0.5 m/sec. We calculate the number of successful samples from a batch of
662 trajectories and average the success across all samples. Through qualitative inspection of
663 the agent, we found that expert behavior occurs at 50% success. Therefore we use a success
664 rate *threshold* of 0.5.
- 665 3. Half-Cheetah Velocity: The half-cheetah agent is successful if the difference between its
666 velocity v_a and the goal velocity v_g is less than 30% of the goal speed $(|v_a - v_g| < 0.3|v_g|)$.
667 We calculate the number of successful samples from a batch of trajectories and average
668 the success across all samples (this is the same as Ant Goal and Ant Direction). We set
669 the success rate threshold to 0.45. The success threshold of 0.45 was chosen by trial and
670 error according to the ability of the learning agent. We found that the highest success PPO
671 could attain, in terms of average success, across these tasks was ~ 0.5 and at 0.45, the agent
672 showcases expert skill on the task.
- 673 4. Metaworld: We use the built-in success threshold from the Metaworld codebase[60]. In the
674 metaworld environments, we care about whether the agent has successfully performed the
675 task during the episode. For this reason, the success value is computed differently than the
676 other environments. We instead average the ratio of trajectories that contained at least one
677 success. We found that agents trained with PPO can reach a success rate of over 0.2 across
678 most tasks while showcasing expert skills in the three environments. We set the success rate
679 threshold to 0.2.

680 **Compute resources** For all our experiments, we used different cloud compute resources. Across
681 these resources, we trained using virtual machines with 8 CPUs and 16 GiB of memory.

B Comparison Training Details

Here we provide training details for the algorithms we compare.

PPO+TL: This algorithm uses a version of PPO that many methods share in this work. This agent trains on each task using only the stochastic gradient objective outlined for PPO. Unlike CoMPS, between tasks PPO+TL does not perform additional meta-learning and instead transfers the network parameters learned this far directly to the next task.

PEARL: The comparison to PEARL uses the provided codebase released with the paper [37]. We updated the PEARL code to change the tasks the PEARL agent trains over. Instead of the original ~ 130 tasks used for training, we refine the agent to train on one task at a time according to the task distributions outlined in the paper. The agent collects 5000 simulation steps between evaluations.

PNC: The PNC algorithm consists of two phases similar to CoMPS. One phase is normal RL which we train using the same parameters and version of PPO that we used for both CoMPS and PPO+TL. The second phase is a *distillation* phase that uses *elastic weight consolidation* to train a different head of the network to imitate the behavior of the current policy on that task while reducing the forgetfulness via a constraint of the change in the output distribution. This distillation phase needs on-policy data; therefore, we devote the last %5 of training to this distillation phase in our experiments. PNC also uses a different network structure than all other methods in the analysis. For PNC the network consists of a policy and a knowledge base with the additional output head for imitation/distillation. Both networks have hidden layers of size [128, 128] and according to [47] the layers in the knowledge base are connected to the layers of the policy.

C RL Training Details

At the end of RL training, we split the data into two portions. These are the \mathcal{D}_k^* and \mathcal{D}_k collections. The number of trajectories of \mathcal{D}_k^* depends on T^{max} for each task, but we fix the batch size to be 5000 samples. We save a portion of the data the RL agent generated to reduce memory costs. We keep a portion of the min of N episodes on RL or the M episodes needed to reach the success threshold. Mathematically, we sample k episodes of data where $k = \min(M, N \cdot 0.05)$. Then we randomly sample 100 trajectories from the k episodes of data. When M is large, this processing avoids keeping around too much data that may lead to memory issues on the computer.

D CoMPS Details

We use a learning rate of 0.005 for each CoMPS meta-learning step, with a total of 25 training steps. For all experiments, we use a two-layered hidden network of sizes [128, 64]. We use a single inner gradient step for all tasks with inner learning rates shown below (Algorithm 1 line 5). For the outer step of CoMPS, we perform imitation learning on \mathcal{D}_k^* for 5 steps on Algorithm 1 line 7.

| Environment | Inner Learning Rate |
|-----------------------|---------------------|
| Ant Goal | 0.005 |
| Ant Direction | 0.005 |
| Half-Cheetah Velocity | 0.005 |
| Metaworld | 0.0025 |

Table 1: CoMPS Hyperparameters

E Additional Results

Additional training results across environments.

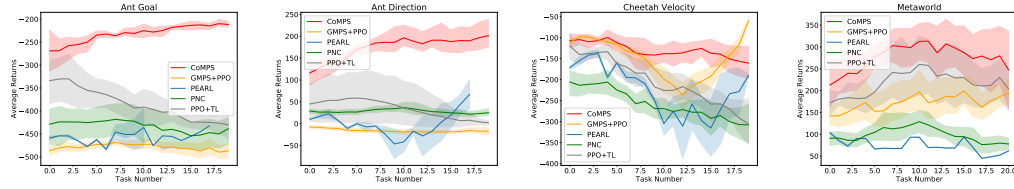


Figure 9: These figures show the average return for each new task. On average CoMPS achieve the highest average return while training over an entire task at a time. Results are computed over 6 sequences of 20 tasks, averaged over 6 random seeds.

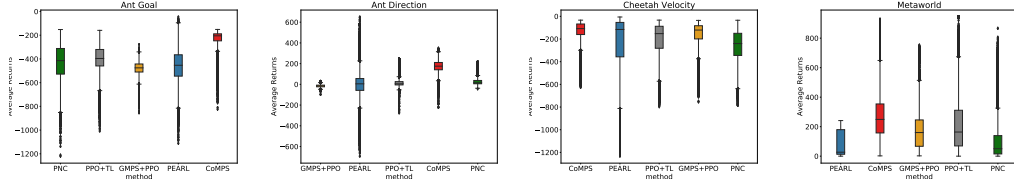


Figure 10: These figures show the average return achieved during the *RL* phase over all tasks. This shows average return the agent achieves if any episode is picked at random from any task. Still, CoMPS performs better than other methods considering this analysis. Results are averaged over 6 sequences of 20 tasks, and 6 random seeds.

E.1 Stationary Task Distributions

In Section 5 we performed experiments to evaluate the performance of CoMPS compared to other methods in terms of learning speed. This performance is measured as the number of learning episodes needed for an agent to *successfully* solve the task. These results can be found in Figure 6 and Figure 12(top). Here we include additional results and metrics for evaluation in addition to these results to get a broader view of the findings. In Figure 9 we show the average return received for the same experiments as in Figure 6. In these results that use uniformly random task distributions, we can see that CoMPS achieves higher average reward while completing the tasks. The CoMPS also improves its ability to achieve high rewards quickly as more tasks are solved. This performance improvement can also be seen in Figure 11(top) that shows the return averaged over tasks, where we average the learning graphs together for each task.

E.2 Non-Stationary Task Distributions

In fig:comparison-avg-ret(left) and Figure 12(bottom-left), we show the average reward of each method for every episode of learning over 20 tasks from a non-stationary distribution. On the right of Figure 7 and Figure 12(bottom) we show the average reward achieved over the same 20 tasks. To provide more aggregate data, we also compute the average reward achieved over all tasks. We provide this information in Figure 10. In this data, we can see that CoMPS obtains a median average return that is higher than all other methods. PEARL, in particular, exhibits a high variance in rewards across tasks, which can also be seen in Figure 7(left). GMPS+PPO, which does not include the additional off-policy importance weighting corrections used in CoMPS, achieves some of the lowest returns across all methods. This further illustrates the importance of the off-policy corrections in CoMPS. We also look at performance per episode averaged over tasks in Figure 11(top). These results also show that CoMPS achieves performance improvements in early episodes across tasks compared to all other methods.

E.3 Video Results

To see videos of the learned policies, see <https://sites.google.com/view/compspaper/home>

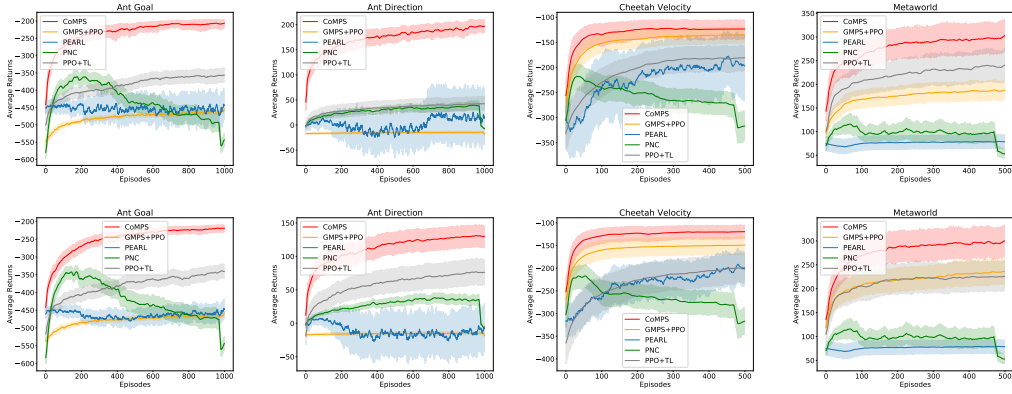


Figure 11: These figures show the average return achieved for each episode in the *RL* phase over all tasks. The top row is for the stationary task distribution and the bottom row for the non-stationary task distribution. These graphs show that CoMPS achieves the highest average return over all other methods when performance is averaged over tasks. Results are averaged over 6 sequences of 20 tasks, and 6 random seeds.

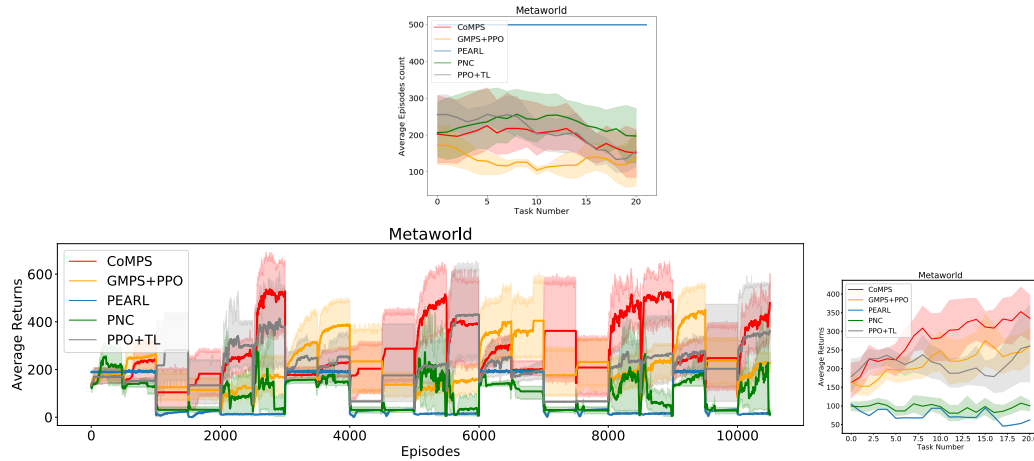


Figure 12: These figures show the results for metaworld including the PEARL comparison that was missing from Figure 6 and Figure 7.

743 F Broader Impacts

744 Our work is a more general optimization scheme that could allow real-world agents and robots to
 745 apply meta-RL better. There may exist a distant automation risk but no more than other papers on RL
 746 or Meta-RL.