Meta-World+: An Improved, Standardized, RL Benchmark

Reginald McLean[*1,2] **Evangelos Chatzaroulas**[*3] **Luc McCutcheon**[3] **Frank Röder**[4] **Zhanpeng He**[5] **Tianhe Yu**[6] **Ryan Julian**[6] **K.R. Zentner**[7] **Jordan Terry**[2] **Isaac Woungang**[1] **Nariman Farsad**[1] **Pablo Samuel Castro**[8,9]

Abstract

Multi-task reinforcement learning challenges agents to master diverse skills simultaneously, and Meta-World emerged as the gold standard benchmark for evaluating these algorithms. However, since the introduction of the Meta-World benchmark there have been numerous undocumented changes which inhibit fair comparison of multi-task and meta reinforcement learning algorithms. This work strives to disambiguate these results from the literature, while also producing an open-source version of Meta-World that has full reproducibility of past results.

1. Introduction

Reinforcement learning (RL) has made significant progress in real-world applications such as the magnetic control of plasma in nuclear fusion (Degrave et al., 2022), controlling the location of stratospheric balloons (Bellemare et al., 2020), or managing a power grid (Yoon et al., 2021). However, each of these RL agents are limited in their abilities as they are only trained to accomplish a single task. While existing benchmarks have pushed progress in specific areas, such as high-fidelity robotic manipulation, or multi-task and meta learning (Wang et al., 2021; Kannan et al., 2021; Bellemare et al., 2013; Nikulin et al., 2023), they tend to be limited to one specific mode of training and evaluation. Evaluating the ability of reinforcement learning agents to both master diverse skills and generalize to entirely new challenges remains a critical bottleneck.

To train RL agents that can accomplish multiple tasks simultaneously, researchers have generally taken one of two approaches. The first is to simultaneously train on multiple tasks with the goal of maximizing accumulated rewards on the same set of tasks. The second approach is to perform meta-learning on a sub-set of tasks where the goal of the RL agent is to learn skills that generalize to a broader set of tasks. Given the significant amount of overlap between these two approaches, Meta-World (Yu et al., 2020b) was proposed to enable effective research on both. Indeed, it has been widely used in foundational multi-task research (Yang et al., 2020; Hendawy et al., 2024; Sun et al., 2022; McLean et al., 2025), as well as single task RL research (Eysenbach et al., 2022; Nauman & Cygan, 2025).

Since its introduction, however, there have been inconsistencies with the versioning of the benchmark, obfuscating effective comparisons between various algorithms, which we empirically demonstrate below. To address these inconsistencies, we re-engineer the benchmark to facilitate research, benchmarking, customization, and reproducibility. Specifically, our contributions are as follows:

1. We perform an empirical comparison of various multi-task algorithms across past reward functions, highlighting the inconsistencies in cross-version comparisons. We then propose insights for future benchmark design based on the empirical results.
2. We add a new task set, in addition to the existing task sets, and introduce the ability for users to create custom multi-task task sets.
3. We upgrade the library to be compatible with the latest Gymnasium API (Towers et al., 2024) & Mujoco Python bindings (Todorov et al., 2012), removing the dependencies on the unsupported packages of OpenAI gym (Brockman et al., 2016) & Mujoco-Py[1].

2. Meta-World

In this section we provide an overview of the Meta-World benchmark. In Appendix A we provide additional information on the different components of Meta-World. Meta-

[1]Toronto Metropolitan University [2]Farama Foundation [3]University of Surrey [4]Hamburg University of Technology [5]Columbia University [6]Google DeepMind [7]University of Southern California [8]Universit de Montral [9]Mila - Quebec Artificial Intelligence Institute. Correspondence to: Reginald McLean <reginald.mclean@torontomu.ca>, Evangelos Chatzaroulas <e.chatzaroulas@surrey.ac.uk>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

[1]https://github.com/Farama-Foundation/Metaworld/

World contains 50 different robotic manipulation tasks for a single Sawyer Robot Arm[2] to solve. Some tasks involve applying force to an object (i.e. *door-close*, *drawer-close*, and *coffee-push*), while some tasks require grasping and manipulating an object (i.e. *pick-place*, and *assembly*), and some combine these two applications of force (i.e. *drawer-open*, *disassembly*). In Appendix A we visualize the included tasks in Meta-World.

2.1. Task Sets

To evaluate algorithms in the multi-task setting the MT10 & MT50 task sets are used. In these sets of 10 and 50 tasks, an algorithm is tasked with learning a distribution of tasks with both parametric and task variations. The parametric variations of each task are the random goal or object locations that can be used, where the task variations are the various types of tasks (i.e. *pick-place*, *reach*, *push*). The agent is evaluated on the same tasks that it is trained on. The evaluation metric for multi-task RL is the mean success rate of an agent across all evaluation tasks.

2.2. Reward Functions

The original publication of Meta-World (Yu et al., 2020b) created a set of dense reward functions. These rewards, which we will refer to as V1, were designed by creating the *pick-place* reward function, and then modifying the *pick-place* reward function to solve a new task. For the *pick-place* reward function, the rewards guide the agent to: reach towards an object, grasp the object, and move the object to a goal location. To create subsequent rewards this structure would be modified for each new task. For example, for the *reach* task, the reward function would be modified to only include the component of rewards for reaching towards the goal. The portion of the reward function for moving towards an object and gripping the object would be removed. However, at some point these V1 rewards were overwritten by a new set of dense rewards, which we will refer to as V2. These rewards were written with fuzzy constraints that allowed for rewards to be in the range $(0, 10)$ and episodic returns across tasks are from a more narrow distribution. To highlight the differences in rewards, in Figure 1 we plot the per-timestep rewards for the *pick-place* task across the V1 and V2 rewards. For V1, the rewards start slightly negative and then reach a maximum reward for success of around 1200. For V2, the rewards start at zero and reach 10 when the task is solved. We include the full reasoning behind designing these reward functions in Appendix A.

[2]Sawyer Arm Information

3. Empirical Results

Due to historical versioning issues in Meta-World, various works have reported results across different reward functions. These results are not comparable, as the two sets of reward functions have different design philosophies and per-task reward scales as illustrated in Figure 1. In Section 3.1, we provide a comparative analysis on a number of multi-task RL methods over the V1 and V2 reward functions. Section 3.2 provides some analysis of our additional task set, MT25.

First, we select and implement the following multi-task RL algorithms: PCGrad (Yu et al., 2020a), multi-task, multi-head, soft actor-critic (MTMHSAC) (Yu et al., 2020b), Soft-Modularization (SM) (Yang et al., 2020), Parameter Compositional (PaCo) (Sun et al., 2022), and Mixture of Orthogonal Experts (MOORE) (Hendawy et al., 2024).

In order to compare the statistical results of each set of experiments we follow recommendations from Agarwal et al. (2021), where we report results over 10 random seeds and use interquartile mean (IQM) to compare results instead of simple averaging. For each graph we plot the IQM performance through training, while also including the 95% confidence interval. Each method is evaluated for 50 evaluation episodes per task, once for each goal location. Hyperparameters for each method are gathered from their respective publications. We implement each method into our open-source baselines code base using JAX (Bradbury et al., 2018) [3].

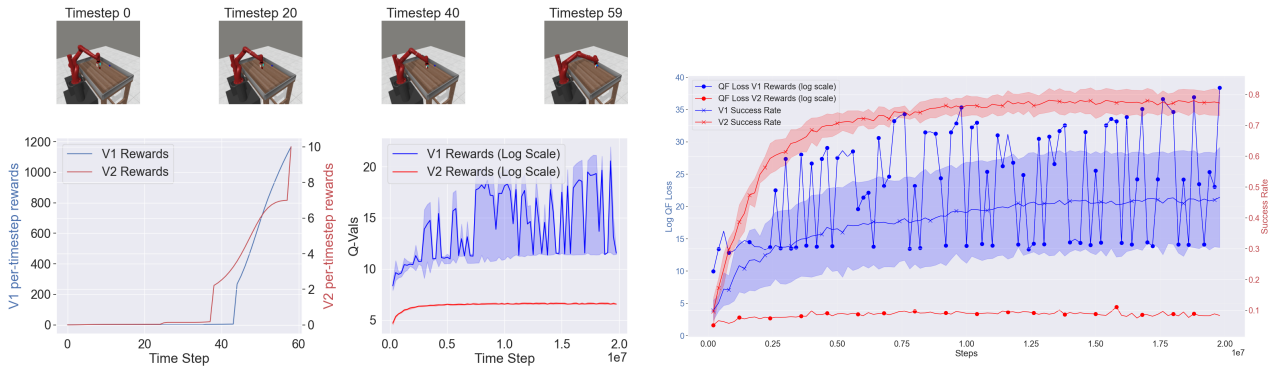
3.1. Multi-task RL results

In this section we outline the MT10 and MT50 results across both the V1 and V2 reward functions, while also corroborating, and refuting, recent results in multi-task RL. In Table 1 we report the performance of the methods that are commonly incorrectly compared due to aforementioned versioning issues. The *Pub* columns indicate values gathered directly from the publication of the method, while the other columns are performance values that we gather using our own implementations of each method.

In Figure 2 we report the IQM learning curves for MT10 and MT50, respectively, trained on each of the selected multi-task RL methods: MTMHSAC, SM, MOORE, PaCo, and PCGrad.

When examining the MT10 results, we find that algorithms generally struggle with the V1 rewards on MT10. This seems to indicate that the V1 rewards are more difficult to optimize as the algorithms have lower performance when compared with the V2 rewards. In the MT10 V2 setting, our results indicate that the V2 reward function is somewhat easier to optimize as the performance of each tested

[3]https://github.com/rainx0r/metaworld-algorithms



(a) (b)

Figure 1: (a) Per-timestep rewards for the *pick-place* task from Meta-World. (Left) The Y-axis shows the scale of the V1 rewards, while the right Y-axis shows the scale of the V2 rewards. The right plot shows the Q-Values per-update batch through training on MT10 (log scale). (b) Q-Function loss and mean success rate of the MTMHSAC on MT10. The left y-axis is the Q-Function loss (log scale), while the right y-axis is the mean success rate. Both blue plots are the V1 rewards, while both red plots are the V2 rewards. Plots with circular points are Q-Function loss, while plots with X points are success rates (with 95% CIs).

algorithm increases compared to the V1 results. Here we find that PCGrad and SM are again the two top performing methods.

Moving on to the MT50 setting, we observe much the same: final performance on the V1 rewards is much lower than on the V2 ones, even more so than on MT10, indicating that the optimization difficulties not only continue as the number of tasks grows but even become exacerbated with more tasks. Therefore, perhaps owing to their respective methods of alleviating said issue, SM and PCGrad continue to outperform the rest of the baselines, though much less so on the V2 rewards, where MOORE is on par with SM.

Through our empirical results and analysis, we find that the implementation of the V2 reward functions does follow the design principle that was intended. In Figure 1(b) we find that when training the MTMHSAC agent on the V1 & V2 rewards, that the Q-Function loss of the V2 trained MTMHSAC agent is much lower than the V1 MTMHSAC, and the V2 rewards lead to a higher success rate overall for the agent. In fact, we find that all tested algorithms perform better on the V2 rewards than the V1 rewards. This seems to show that in multi-task RL the capability of the Q-Function to model state-action values is tied to the overall success rate of the agent, which agrees with recent work (McLean et al., 2025).

3.2. Results over various task set sizes

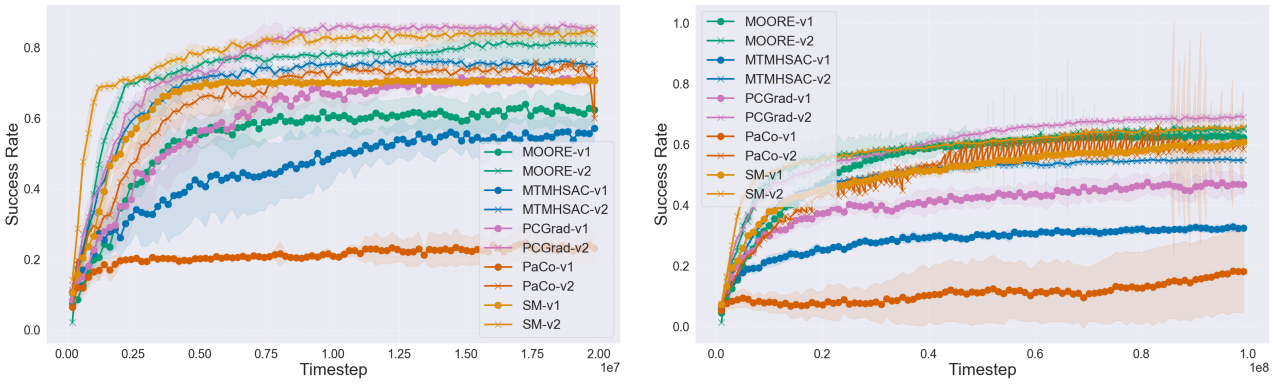
Finally, in Figure 3, we explore the effects of introducing an additional task set to the Meta-World benchmark. We discuss the construction of MT25 in Appendix B. The introduction of MT25 and customizable task sets offers significant practical advantages for researchers. Conducting experiments on MT50 can be computationally expensive, where MT25 provides a middle ground that reduces computational costs by approximately 50% compared to MT50 while still offering more robust insights than the smaller MT10 benchmark. Our own experiments had walltimes of: MT10 $\sim$ 6 hours, MT25 $\sim$ 12 hours, and MT50 $\sim$ 25 hours on an AMD Epyc 7402 24-Core Processor with an NVIDIA A100 PCI GPU. This efficiency allows researchers to conduct preliminary experiments and algorithm comparisons more rapidly, reserving full MT50 evaluations for finalized methods.

4. Overall Benchmark Improvements

Through this update process, we have made several important updates and upgrades to Meta-World. The following sections highlights what we deem to be the most important ones.

4.1. Gymnasium Integration

By aligning Meta-World with the Gymnasium standard rather than maintaining its custom environment implementation, we enable researchers to leverage the full ecosystem of Gymnasium tools and infrastructure. This integration is particularly valuable for performance optimization through Gymnasium's vectorized environment capabilities. The AsyncVectorEnv wrapper, for example, explicitly adds support for parallel environment stepping across multiple CPU cores, increasing throughput for multi-task experi-



(a) (b)

Figure 2: Results on (a) MT10, and (b) MT50

ments that require interaction with numerous environments simultaneously.

Beyond performance benefits, this standardization grants immediate access to Gymnasium's extensive library of wrappers for observation processing, reward shaping, and monitoring - eliminating the need for custom implementations of these common functionalities. Furthermore, this alignment reduces the learning curve for new users who are already familiar with the Gymnasium API, allowing researchers to focus on algorithm development rather than environment interfacing details. This integration represents not just a technical improvement but a significant step toward broader accessibility and standardization in reinforcement learning research.

4.2. Quality of Life Improvements

The streamlined benchmark creation process significantly reduces the barrier to entry for new users. Previously, environment instantiation required detailed knowledge of Meta-World's internal architecture and multiple configuration steps. With the new *gym.make* interface, users can create environments using a single, intuitive command, following the familiar Gymnasium paradigm. This simplification not only saves development time but also reduces the likelihood of configuration errors. In Figure 8 we show how we have simplified the environment creation process.

4.3. Full Reproducibility of Past Results

Maintaining backward compatibility while introducing these improvements ensures that researchers can reproduce and build upon previous Meta-World results. We have carefully preserved the core functionality and task definitions from earlier versions while adding new capabilities. This approach allows researchers to verify their implementations against established benchmarks and directly compare their results with previous work. The updated code base includes comprehensive documentation of any minor variations in task definitions or reward structures, ensuring transparency in how modern implementations relate to historical results.

These improvements collectively modernize Meta-World while preserving its value as a research benchmark. The combination of standardized interfaces, improved usability, and maintained reproducibility positions Meta-World as an even more valuable tool for reinforcement learning research.

5. Conclusion

In this work we highlighted a discrepancy in the multi-task literature due to issues with the software versioning of the Meta-World benchmark. Through empirical evaluation, we demonstrated that inconsistent reward functions led to incorrect conclusions about algorithm performance, with multi-task RL methods showing significant sensitivity to reward scaling differences.

This work highlights the need for explicit versioning of software benchmarks that researchers should report in their works. Had this versioning been in place throughout the history of Meta-World, the discrepancies between results likely would not have happened. In addition to software versioning, we have also highlighted the need for better practices when making large changes to an existing benchmark that may affect results. Lastly, we find that simply reporting results from a previous work, rather than running the method (through one's own implementation, or open sourced code), likely also contributed to these issues with Meta-World. Thus, we believe that users should be running their own baselines rather than copying numbers from previous works.

Due to the issues that we have highlighted in this work, we argue that benchmark development should prioritize transparent versioning and explicit documentation of design decisions that affect performance. Beyond addressing versioning issues, next-generation multi-task benchmarks should explore greater task diversity, compositional complexity, and cross-embodiment transfer challenges to better differentiate algorithmic advances. By building standardized evaluation protocols with these principles in mind, the community can ensure that reported improvements reflect genuine algorithmic progress rather than artifacts of implementation differences or benchmark design variations.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=uqv8-U4lKBe.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: an evaluation platform for general agents. *J. Artif. Int. Res.*, 47(1):253279, May 2013. ISSN 1076-9757.

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2939-8. URL https://doi.org/10.1038/s41586-020-2939-8.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de las Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M., Noury, S., Pesamosca, F., Pfau, D., Sauter, O., Sommariva, C., Coda, S., Duval, B., Fasoli, A., Kohli, P., Kavukcuoglu, K., Hassabis, D., and Riedmiller, M. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, February 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04301-9. URL https://www.nature.com/articles/s41586-021-04301-9. Number: 7897 Publisher: Nature Publishing Group.

Eysenbach, B., Zhang, T., Levine, S., and Salakhutdinov, R. Contrastive learning as goal-conditioned reinforcement learning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=vGQiU5sqUe3.

Hendawy, A., Peters, J., and D'Eramo, C. Multi-task reinforcement learning with mixture of orthogonal experts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=aZH1dM3GOX.

Kannan, H., Hafner, D., Finn, C., and Erhan, D. Robodesk: A multi-task reinforcement learning benchmark. https://github.com/google-research/robodesk, 2021.

McLean, R., Chatzaroulas, E., Terry, J. K., Woungang, I., Farsad, N., and Castro, P. S. Multi-task reinforcement learning enables parameter scaling. In *Reinforcement Learning Conference*, 2025. URL https://openreview.net/forum?id=eBWwBIFV7T.

Nauman, M. and Cygan, M. Decoupled policy actor-critic: Bridging pessimism and risk awareness in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19633–19641, 2025.

Nikulin, A., Kurenkov, V., Zisman, I., Sinii, V., Agarkov, A., and Kolesnikov, S. XLand-minigrid: Scalable meta-reinforcement learning environments in JAX. In *Intrinsically-Motivated and Open-Ended Learning Workshop @NeurIPS2023*, 2023. URL https://openreview.net/forum?id=xALDC4aHGz.

Sun, L., Zhang, H., Xu, W., and Tomizuka, M. Paco: Parameter-compositional multi-task reinforcement learning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=LYXTPNWJLr.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Wang, J. X., King, M., Porcel, N. P. M., Kurth-Nelson, Z., Zhu, T., Deck, C., Choy, P., Cassin, M., Reynolds, M., Song, H. F., Buttimore, G., Reichert, D. P., Rabinowitz, N. C., Matthey, L., Hassabis, D., Lerchner, A., and Botvinick, M. Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=eZu4BZxlRnX.

Yang, R., Xu, H., Wu, Y., and Wang, X. Multi-task reinforcement learning with soft modularization. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/32cfdce9631d8c7906e8e9d6e68b514b-Abstract.html.

Yoon, D., Hong, S., Lee, B.-J., and Kim, K.-E. Winning the l2{rpn} challenge: Power grid management via semi-markov afterstate actor-critic. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=LmUJqB1Cz8.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5824–5836. Curran Associates, Inc., 2020a. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/3fe78a8acf5fda99de95303940a2420c-Paper.pdf.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020b.

Yu, T., Quillen, D., He, Z., Julian, R., Narayan, A., Shively, H., Bellathur, A., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. URL https://arxiv.org/abs/1910.10897.

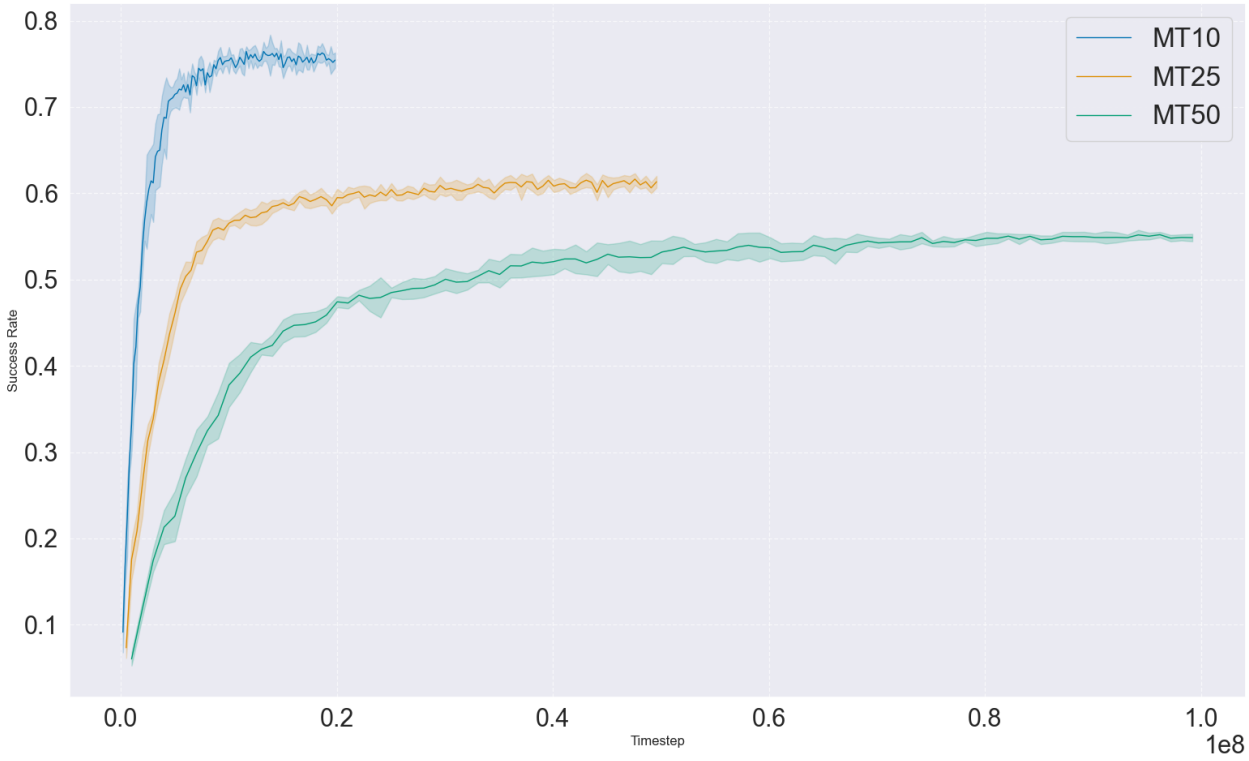


Figure 3: Effects of various task set sizes on the performance of the MTMHSAC agent.

A. Meta-World

Meta-World is a collection of robotic manipulation tasks with a shared observation & action space designed to learn multi-task or meta-RL policies on a collection of tasks. In previous works, such as Yang et al. (2020), there is some discussion of 'fixed' and 'conditioned' versions of Meta-World. This distinction does not exist in the current version of Meta-World, though it may have in earlier versions of Meta-World.

A.1. Additional Quality of Life Improvements

A.1.1. Quality of Life Improvements

Figure 8 shows the new streamlined benchmark creation process which significantly reduces the barrier to entry for new users. Previously, environment instantiation required detailed knowledge of Meta-World's internal architecture and multiple configuration steps. With the new *gym.make* interface, users can create environments using a single, intuitive command, following the familiar Gymnasium paradigm.

A.1.2. Full Reproducibility of Past Results

Maintaining backward compatibility while introducing these improvements ensures that researchers can reproduce and build upon previous Meta-World results. We have carefully preserved the core functionality and task definitions from earlier versions while adding new capabilities. This approach allows researchers to verify their implementations against established benchmarks and directly compare their results with previous work.

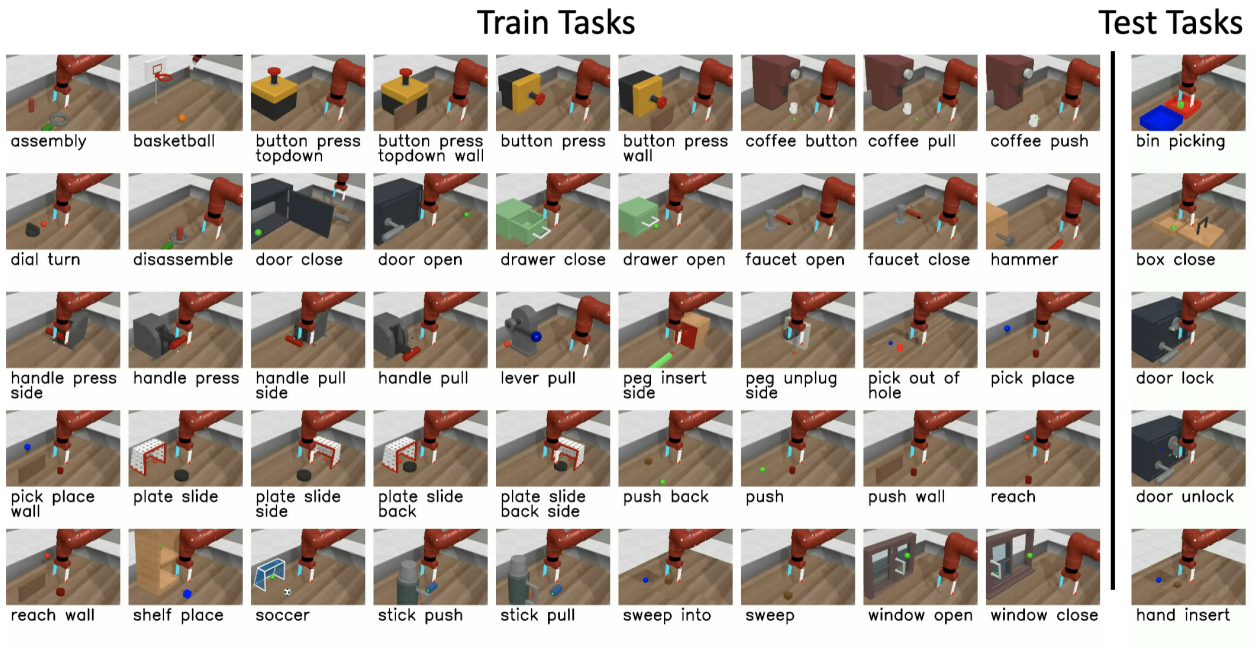


Figure 4: Tasks available within Meta-World. Image from Yu et al. (2021).

A.2. Tasks

A.3. Reward function

For each of the 50 available tasks in Meta-World, there are 2 dense reward functions. The first reward function, V1, was used to produce the results in (cite CoRL paper). The second reward function, V2, was used to produce the results in (cite Meta-World arxiv paper). We believe that by exposing both the V1 and V2 reward functions to the user, that users can more accurately compare the results of their algorithms to other published results. This would also allow for users to evaluate their algorithms on variations of Meta-World benchmarks that may show how well their methods work in various settings.

A.4. Observation & Action Spaces

The observation space and action spaces are inherited from the Gymnasium standard for continuous spaces. The action space is a 4-tuple where the first three elements of the 4-tuple are the desired end-effector displacement, and the last element is the position of the gripper fingers.

The observation space is designed to be shared across all 50 available tasks in Meta-World. Thus, they must contain information for all of the different task scenarios. In order to be used across all tasks, each observation in Meta-World is 39-dimensional. Positions (zero indexed, inclusive) in the observation represent the following attributes:

- $[0:2]$: the XYZ coordinates of the end-effector
- $[3]$: a scalar value that represents how open/closed the gripper is
- $[4:6]$: the XYZ coordinates of the first object
- $[7:10]$: the quaternion describing the spatial orientations and rotations of object #1
- $[11:13]$: the XYZ coordinates of the second object
- $[14:17]$: the quaternion describing the spatial orientations and rotations of object #2

Both of the objects XYZ coordinates and quaternions can exist (i.e. in the *coffee-push* task), or if only one object exists the second object XYZ coordinates and quaternion are zeroed out. List A.4 only outlines an observation of size 18. The

observation at time t is stacked with the observation from time $t-1$, and the XYZ coordinates of the goal, to complete the 39-dimension observation vector. Meta-World is thus a goal-conditioned task. In multi-task reinforcement learning the goal is visible to the agent, while in meta-reinforcement learning the goal is zeroed out.

A.5. Tasks & Task-Sets

```
import gymnasium as gym
import metaworld
# this registers the Meta-World environments with Gymnasium

# create a MT1 environment object
envs = gym.make('Meta-World/MT1', env_name='reach-v3', seed=42)

# create a MT10 environment object
envs = gym.make('Meta-World/MT10', vector_strategy='sync', seed=42)

# create a MT50 environment object
envs = gym.make('Meta-World/MT50', vector_strategy='sync', seed=42)

# create a custom benchmark environment object
envs = gym.make('Meta-World/MT-custom', env_names=['env1-v3', 'env2-v3', ..., 'envN-v3'],
                                        vector_strategy='sync', seed=42)
```

Figure 5: Multi-task environment creation examples. Note that multi-task algorithms are evaluated on the training environments.

```
import gymnasium as gym
import metaworld
# this registers the Meta-World environments with Gymnasium

# create ML1 environment objects
train_envs = gym.make('Meta-World/ML-1-train', env_name='reach-v3', seed=42)
test_envs = gym.make('Meta-World/ML-1-test', env_name='reach-v3', seed=42)

# create ML10 environment objects
train_envs = gym.make('Meta-World/ML10-train', vector_strategy='sync', seed=42)
test_envs = gym.make('Meta-World/ML10-test', vector_strategy='sync', seed=42)

# create a ML45 environment object
train_envs = gym.make('Meta-World/ML45-train', vector_strategy='sync', seed=42)
test_envs = gym.make('Meta-World/ML45-test', vector_strategy='sync', seed=42)

# create a custom ML benchmark environment object
train_envs = gym.make('Meta-World/ML-custom', env_names=['env1-v3', 'env2-v3', ..., 'envN-
                                        v3'], vector_strategy='sync', seed=42)
test_envs = gym.make('Meta-World/ML-custom', env_names=['env1-v3', 'env2-v3', ..., 'envN-
                                        v3'], vector_strategy='sync', seed=42)
```

Figure 6: Meta-learning environment creation examples. Note that meta-learning tasks have a distinction between training and testing environments.

B. MT25 and ML25 Construction

To create a new task set, we used the results from training on MT50. To create the training task set of 25 tasks, we select 12 tasks that are solved in MT50 and 13 tasks that are not solved.

```python
import gymnasium as gym
import metaworld
# this registers the Meta-World environments with Gymnasium
from metaworld.evaluation import evaluation, metalearning_evaluation, Agent,
                                                    MetaLearningAgent

# Multi-task
# Explicitly subclassing is optional but recommended
class MyAgent(Agent):
    def eval_action(self, observations):
        ... # agent action selection logic

agent = MyAgent()
envs = gym.make('Meta-World/MT10', vector_strategy='sync', seed=42)
mean_success_rate, mean_returns, success_rate_per_task = evaluation(agent, envs)

# Meta-Learning
# Explicitly subclassing is optional but recommended
class MyMetaLearningAgent(MetaLearningAgent):

    def eval_action(self, observations):
        # agent action selection logic
        ...

    def adapt_action(self, observations):
        # agent action selection logic
        ...
        # this can also return values needed for adaptation like logprobs

    def adapt(self, rollouts) -> None:
        # Adaptation logic here, this should mutate the agent object itself
        ...


agent = MyMetaLearningAgent()
test_envs = gym.make('Meta-World/ML10-test', vector_strategy='sync', seed=42)
mean_success_rate, mean_returns, success_rate_per_task = metalearning_evaluation(agent,
                                                    test_envs)
```

Figure 7: Sample code for evaluating agents on Meta-World.

Algorithm	Pub MT10	MT10 V1	MT10 V2	Pub MT50	MT50 V1	MT50 V2
SM	71.8	**71.4**	84.9	61.0	**60.6**	65.8
PaCo	85.4	26.2	**73.6**	57.3	18.6	**58.4**
MOORE	88.7	61.4	**83.2**	72.9	61.2	**65.3**

Table 1: Comparing results from a recent multi-task RL publication (Hendawy et al., 2024), to the results we empirically validate across the V1 & V2 reward functions. Pub MT10 and Pub MT50 are from Hendawy et al. (2024), while the V1 and V2 columns are of our own implementations. Bolded results indicate the reward function that was used to produce the result for that specific method.

```
import metaworld
import random
from random import choice

random.seed(42)
mt10 = metaworld.MT10(seed=42)
training_envs = []

for name, env_cls in
mt10.train_classes.items():
  env = env_cls()
  task = choice([task for task in
mt10.train_tasks if task.env_name ==
name])
  env.set_task(task)
  training_envs.append(env)
for env in training_envs:
    action = env.action_space.sample()
    next_ob, r, termination, truncation,
info = env.step(action)
```

```
import gymnasium as gym
import metaworld

envs = gym.make('Meta-World/MT10',
vector_strategy='sync', seed=42)

actions = envs.action_space.sample()
next_obs, rewards, terminations,
truncations, infos = envs.step()
```

Python imports

Environment Creation

Applying a single action to each environment

Figure 8: (Left) Old environment creation process, (Right) New environment creation process.