Technical Appendix

In this supplementary material, we detail the experimental approach and the configuration of our experiments. We depict an overview of used hyper-parameters and review differences between the compared code bases. We include the learning curves of the experiments (A-C) for all eight environments. Finally, we conduct an extended analysis that contributes to our discussion from the main paper.

Experimental Setup and Hyper-parameters

For all experiments - if not otherwise specified in the experimental setups of Exp. A-D - we used the hyper-parameters listed in Table 1 as default. The network structure for both policy and value network is a multi-layer perceptron with two hidden layers each consisting of 64 neurons and followed by tanh non-linearities. Weights are not shared between value and policy network. This configuration is based on the one used in (Henderson et al. 2018).

Agent act on-policy such that they sequentially interact B-times with the environment and, then, performs M policy iterations based on this experience batch before generating a new experience batch of size B. This modus operandi is denoted as on-policy learning within the RL community.

During the training, actions are generated over the Gaussian distribution $\mathcal{N}(\pi(x), \epsilon I)$, where $\pi(x)$ is the output vector of the policy representing the mean of the Gaussian and ϵI is the co-variance matrix defined by the scalar ϵ and the identity I. For evaluation purposes, we deactivate the exploration noise such that the stochastic policy function becomes a deterministic function of the state, i.e. $u = \pi(x)$ instead of $u \sim \mathcal{N}$.

For all experiments, we rely upon a single learner setup, i.e. each policy trained in an individual process and does not share information with other processes (this setup is in contrast to distributed or multiple actor setup). We used a Threadripper 3970X CPU, which is capable to process 64 threads in parallel. We provide the source code as repository at https://github.com/successful-ingredients-paper.

Hyper-parameter	Value
Backtracking steps (only TRPO)	10
Batch-size B	32000
Conjugate Gradient damping	0.1
Conjugate Gradient iterations	10
Discount factor γ	0.99
Entropy co-efficient	0.01
Exploration noise σ	0.5
GAE factor λ	0.95
Gradient scaling (L2 norm)	0.5
Learning rate policy network α_π	(*)
Learning rate value network	1.0e-3
Mini-batch size value network	64
Number of mini-batch updates (value network)	80
Optimizer (policy & value network)	Adam
Optimizer Denominator ϵ	1e-08
Target KL divergence δ	(*)
Total time steps	10,000,000
Train iterations of policy network M	(*)
V-trace truncation threshold $\bar{c}$	1.0
V-trace truncation threshold $\bar{\rho}$	1.0
Weight initialization gain κ	$\sqrt{5}$
Weight initialization scheme	Kaiming Uniform

Table 1: Default hyper-parameters used in all experiments. The values marked with asterix are determined through grid search (see approach in the respective experiment section).

Experiments A: Code-level ingredients.

We searched over learning rates for the policy network $\alpha_\pi \in [0.0001, 0.00025, 0.0005, 0.001]$ and numbers of policy train iterations $M \in [10, 20, 40, 80]$. Each (α_π, M)-pair was evaluated over 16 independent random seeds, while each of the seven code-level ingredients was randomly applied by a probability of 50%.

In total, the results were obtained through 256 independent random seeds for each benchmark tasks. The final performance was measured after 10M environment interactions while exploration noise being disabled. Numerical scores were calculated by $(J_\iota^+ - J_\iota^-)/(J_\iota^- - J_\chi)$, where J_ι^+ and J_ι^- denote the policy performance when ingredient ι is on and off, respectively, and

J_χ is the performance of the random policy χ (see Table 3 for the scores of the random policy χ). The learning curves of all environments are depicted in the respective subsection: Experiments A. Code-level ingredients.

Now, we elaborate on the code-level ingredients and their specifics. Opposed to the commonly applied clipping of values in some code-bases, we do not perform a clipping for observation and advantage standardization, and neither for the reward scaling. We make use of the L2 norm for gradient scaling, which is only applied to the policy network. The value network is untouched by all ingredients for all experiments, except for parameter-sharing studied in Experiment C. If an annealing strategy is enabled for the policy learning rate or the exploration noise, we decay it linearly over the course of learning. The policy learning rate α_π is annealed to zero, whereas the exploration noise standard deviation σ is decayed from 0.5 to 0.2 during the training.

Table 2 shows the numerical values of the code-level experiments, which are visualized in Figure 1 of the main paper. Note that we could not achieve learning progress at the Kuka task without advantaged variance reduction methods and, thus, we ignored it in Figure 1, but present results in the next sections. The full learning curves can be seen in the supplemental.

Ingredient	HalfCheetah	Ant	Hopper	Walker2D	Humanoid	Reacher	Pusher	Kuka
Observation Stand.	33.77	57.68	82.38	39.93	17.16	-1.54	10.74	-26.14
Advantage Stand.	-4.89	-14.00	-2.14	-2.76	0.77	-8.11	-13.14	-8.53
Reward Scaling	8.00	9.90	17.66	3.46	-3.62	-4.15	-20.62	14.36
Gradient Scaling	-2.70	-18.96	-3.75	4.18	-0.51	0.73	-29.21	10.48
Entropy Bonus Term	-1.58	-11.58	5.84	9.65	-1.29	-0.95	16.35	0.23
Exploration Noise An.	-5.63	-20.64	-17.23	-11.94	21.86	6.58	0.53	-0.25
Learning Rate An.	15.60	40.05	29.31	54.39	16.07	11.56	48.18	1.23

Table 2: Impact of code-level ingredients on the policy performance in percent. In each run, an ingredient is randomly applied by a chance of fifty percent. The numbers are averaged over 256 independent random seeds for each environment. Note that these numbers are generated on the basis of the IWPG objective without algorithmic ingredients such as trust-regions, which are later added in Experiment B. These results are visualized in Figure 1 of the paper's main part. Note that we excluded the Kuka task because we could not achieve a performance gain without GAE or V-trace.

Environment	Score
HalfCheetahBullet	-1265.58
AntBullet	369.45
HopperBullet	20.01
Walker2DBullet	16.74
HumanoidBullet	-33.16
ReacherBulletEnv	-12.77
PusherBulletEnv	-210.11
ThrowerBulletEnv	-717.89
KukaBulletEnv	-2626.50

Table 3: Scores of a random policy that samples uniformly from the action space.

Experiments B: Algorithmic ingredients.

For the experiments about the effect of algorithmic ingredients, we conducted runs over all possible combinations of trust-region enforcement and variance reduction methods. The best score for each combination was determined through a grid search over four independent runs. In particular, we searched over the learning rates $\alpha_\pi \in [0.0001, 0.00025, 0.0005, 0.001]$ and numbers of policy iterations $M \in [10, 20, 40, 80]$ for IWPG, ESC, PPO; whereas for NPG and TRPO, we searched over the maximum KL divergence $\delta \in [0.001, 0.0025, 0.005, 0.0075, 0.01, 0.025, 0.05, 0.075, 0.1]$.

As a reference for comparison, we took the plain IWPG objective as used in experiment A with plain advantage estimation, and the activated code-level ingredients observation standardization, linear learning rate annealing, but without the use of a trust-region enforcement. We additionally applied reward scaling for the locomotion tasks but deactivated reward scaling for the manipulation tasks. The numerical scores can be taken from Table 4.

For each possible combination between trust-region enforcement and variance reduction method, a grid search over learning rates and number of policy iterations was performed and only the best return J averaged over four independent runs is depicted, i.e. $\max \text{mean}(J) - \text{std}(J)$. The numerical values are displayed in Table 4.

HalfCheetah	IWPG	ESC	NPG	TRPO	PPO
Plain	1793±71	1957±120	2354±223	2200±123	**2583±130**
GAE	2290±116	2316±184	**3158±87**	2977±206	3061±118
V-trace	2281±288	2421±63	2879±169	2699±171	**2997±196**
Ant	IWPG	ESC	NPG	TRPO	PPO
Plain	2445±82	2589±144	2950±155	2865±185	**3042±95**
GAE	3004±196	3202±88	**3609±56**	3288±37	3472±123
V-trace	2808±65	2830±65	3136±177	**3221±135**	3010±189
Hopper	IWPG	ESC	NPG	TRPO	PPO
Plain	1934±123	1896±152	**2381±87**	2309±160	2331±77
GAE	2274±55	2304±33	2514±11	2506±35	**2729±84**
V-trace	1895±162	2039±292	2363±56	2485±12	**2677±47**
Walker2D	IWPG	ESC	NPG	TRPO	PPO
Plain	869±37	823±16	789±49	838±82	**854±19**
GAE	1345±321	939±6	2571±84	**2614±100**	2286±226
V-trace	912±42	950±33	1889±253	**2181±201**	1894±237
Humanoid	IWPG	ESC	NPG	TRPO	PPO
Plain	133±5	138±9	158±10	155±6	**159±8**
GAE	125±3	142±13	890±626	355±110	**2376±775**
V-trace	137±12	148±16	165±2	174±5	**199±28**
Reacher	IWPG	ESC	NPG	TRPO	PPO
Plain	0.33±1.29	1.15±1.22	15.73±0.69	19.30±2.59	**20.71±1.43**
GAE	16.91±0.83	16.26±1.64	20.26±0.98	20.63±0.84	**22.63±2.06**
V-trace	14.76±2.27	16.70±1.57	19.95±0.66	20.62±0.44	**21.01±1.60**
Pusher	IWPG	ESC	NPG	TRPO	PPO
Plain	-136.79±3.43	-136.83±3.38	-144.08±2.97	-141.46±5.10	**-134.50±2.36**
GAE	-139.44±3.93	-142.14±3.24	-142.81±2.28	-143.30±2.75	**-141.21±1.51**
V-trace	-148.63±2.41	**-139.06±1.18**	-148.98±0.31	-145.87±0.52	-144.69±1.23
Kuka	IWPG	ESC	NPG	TRPO	PPO
Plain	-3191±511	-2927±458	-2711±535	-2977±685	**-2705±235**
GAE	-2189±61	-2157±37	-2143±27	**-1962±110**	-1842±246
V-trace	-2183±160	-2180±129	-2074±6	-1956±101	**-1833±118**

Table 4: Performance overview of trust-regions and advantage estimation methods (best result of each row is depicted in bold font). Note that except for IWPG and ESC with plain advantage estimation the algorithms failed to learn on the Humanoid task. An interesting observation is that ESC does on average better than IWPG, which underlines the importance of trust-region enforcement (ESC is identical to IWPG except that policy updates are stopped as soon the KL distance criterion is exceeded).

Experiments C: Structural ingredients

Regarding the stage-wise analysis, we built on the findings of previous experiments and used IWPG with observation standardization and learning rate annealing as well as reward scaling (only for locomotion tasks) as code-level ingredients and applied GAE for advantage variance reduction. We performed a grid search over a 4×4 combination of policy learning rate α_π and number of train iterations M and accumulated all scores over four independent seeds. The results are illustrated as cumulative distribution functions (CDF) where each graph line holds 64 independent individual runs (see Figures 2-8). For improved visuals, the graphs are smoothed with a hamming filter of size 17. As learning rates we used $\alpha_\pi \in [0.001, 0.0025, 0.005, 0.01]$ for SGD and $\alpha_\pi \in [0.0001, 0.00025, 0.0005, 0.001]$ for Adam and RMSProp. We used the same number of policy training iterations $M \in [10, 20, 40, 80]$ for all three optimizers. As configuration for the optimizers, we used:

1. **Adam** with $\epsilon = 10^{-8}$ as term added to the denominator, with coefficients for computing running averages $\beta = (0.9, 0.999)$, no weight decay
2. **RMSprop** with $\epsilon = 10^{-8}$ as term added to the denominator, smoothing constant $\beta = 0.99$, no momentum, no weight decay, not centered
3. **SGD** without momentum, no weight decay, no Nesterov momentum

Regarding the initialization schemes, we initialized the bias parameters with zeros. The weight matrices are initialized according to:

1. **Kaiming Uniform** with gain $\kappa = \sqrt{5}$
2. **Glorot** with gain $\kappa = 1$
3. **Orthogonal** with gain $\kappa = \sqrt{2}$

We denote rankings to determine the best setup over all tasks (see Tables 5 - 8). The scores are determined by the average over the CDF plot.

	HalfCheetah	Ant	Hopper	Walker2D	Humanoid	Reacher	Pusher	Kuka	Mean Rank
SGD	3	3	3	3	3	3	3	3	3.000
Adam	2	2	2	1	2	2	2	2	1.875
RMSprop	1	1	1	2	1	1	1	1	1.125

Table 5: Ranking of optimizers.

	HalfCheetah	Ant	Hopper	Walker2D	Humanoid	Reacher	Pusher	Kuka	Mean Rank
Kaiming	1	1	1	2	1	2	1	3	1.500
Glorot	2	2	3	1	2	3	3	2	2.250
Orthogonal	3	3	2	3	3	1	2	1	2.250

Table 6: Ranking of initialization schemes.

	HalfCheetah	Ant	Hopper	Walker2D	Humanoid	Reacher	Pusher	Kuka	Mean Rank
True	2	2	2	2	2	2	1	2	1.875
False	1	1	1	1	1	1	2	1	1.125

Table 7: Ranking of parameter sharing.

	HalfCheetah	Ant	Hopper	Walker2D	Humanoid	Reacher	Pusher	Kuka	Mean Rank
1e-08	1	3	1	2	4	4	1	1	2.125
1e-07	4	1	2	4	2	2	4	3	2.750
1e-06	3	4	3	3	3	1	2	4	2.875
1e-05	2	2	4	1	1	3	3	2	2.250

Table 8: Ranking of Adam's ϵ.

Experiments C. All CDF Plots

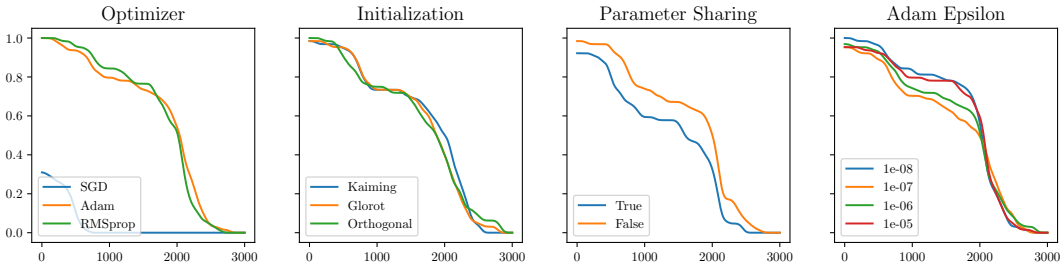

Figure 1: HalfCheetah.

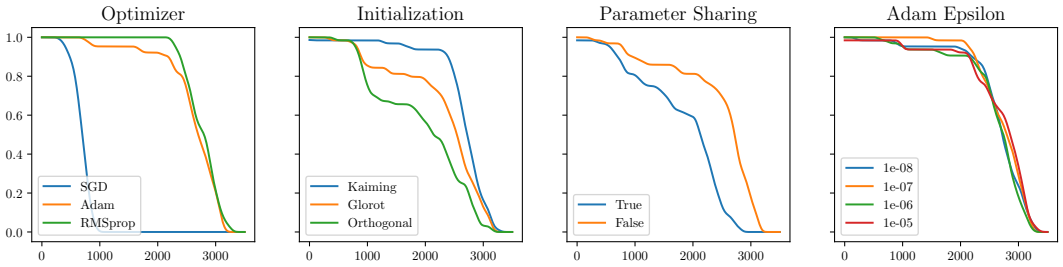

Figure 2: Ant.

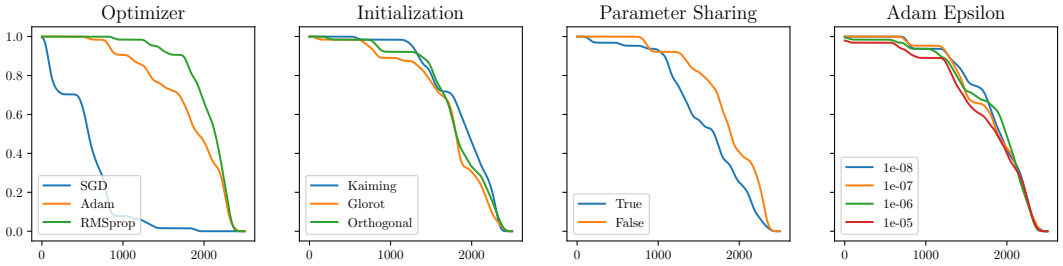

Figure 3: Hopper.

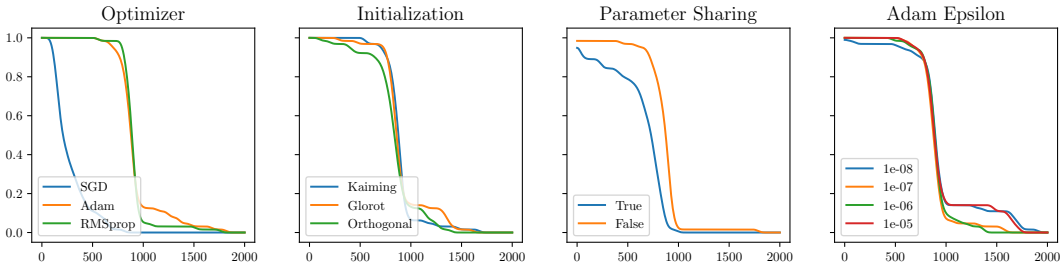

Figure 4: Walker2D.

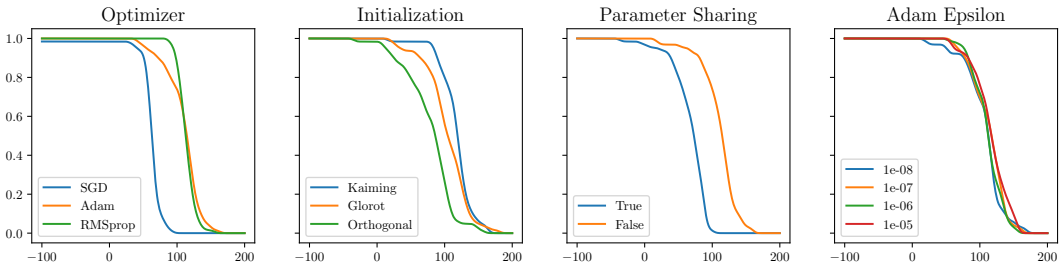

Figure 5: Humanoid.

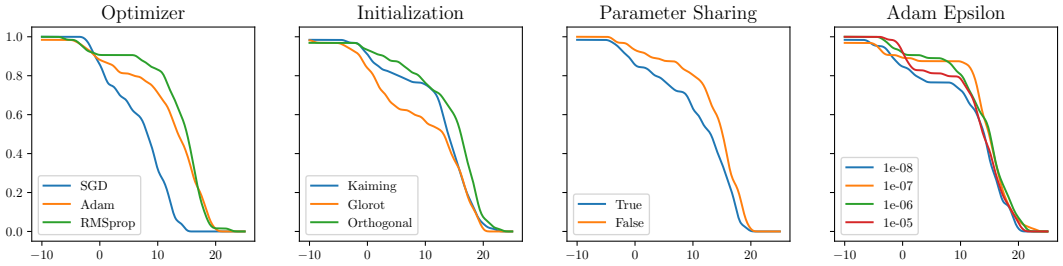

Figure 6: Reacher.

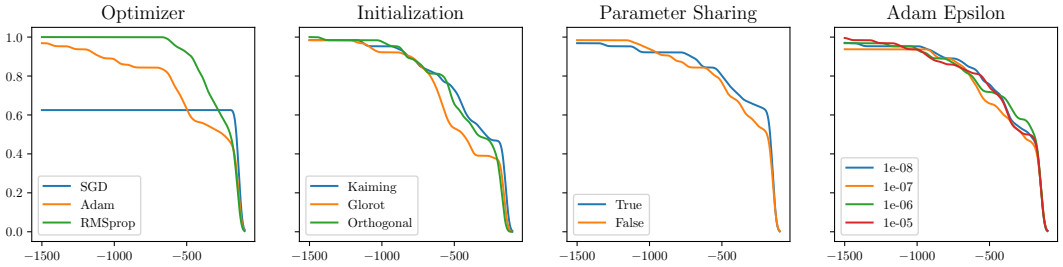

Figure 7: Pusher.

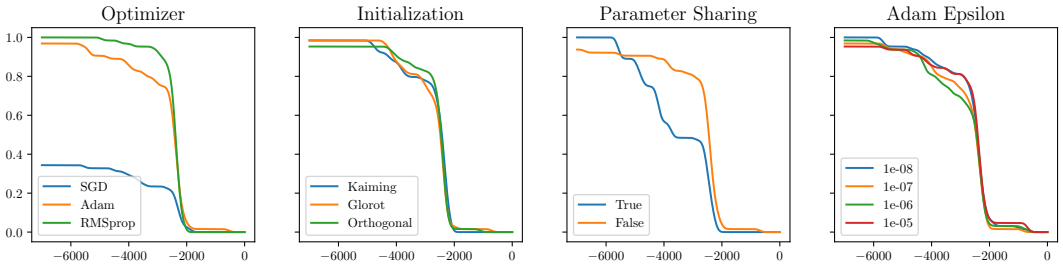

Figure 8: Kuka.

Experiments D: Identifying the Minimal Setup

We used the IWPG objective enhanced with standardized observations and learning rate annealing as the minimal setup. Reward scaling was applied in locomotion tasks but not for manipulation tasks. We tested both RMSprop and Adam as policy optimizers together with the Kaiming initialization scheme. We compared this composition of ingredients to the popular repository OpenAI's Baselines (Dhariwal et al. 2017).

For performance comparison, we conducted a grid search over learning rates $\alpha_\pi \in [1e-4, 2.5e-4, 5.0e-4, 1e-3]$ for the Adam/RMSprop optimizer and the numbers of policy iterations $M \in [10, 20, 40, 80]$ for IWPG and PPO; whereas for TRPO we searched over maximum trust-region sizes $\delta \in [1e-3, 2.5e-3, 5e-3, 7.5e-3, 0.01, 0.025, 0.05, 0.075, 0.1]$ and selected the best score averaged over four independent seeds. To give an overview, we compare selected ingredients for these experiments in Table 9.

Ingredient	IWPG	PPO	TRPO
Obs. stand	✓	✓	✓
Rew. Scaling	[✓, ✗]*	✓	✓
Adv. Stand	✗	✓	✓
Gradient Scaling	✗	L2	✗
Entropy Term	✗	✗	✗
Exploration Noise An.	✗	✗	✗
Learning Rate An.	✓	✓	N/A
Target KL	✗	✗	✓
Optimizer	[Adam, RMSprop]	Adam	N/A
Kernel Initilization	Kaiming	Orthogonal	Orthogonal
Parameter Sharing	✗	✗	✗

Table 9: Overview of ingredients used for code base comparison. The used hyper-parameters are identical to the ones from Table 1.

(*) For IWPG, reward scaling was applied to the locomotion but not to the manipulation tasks.

Learning Curves

Experiments A. Code-level ingredients

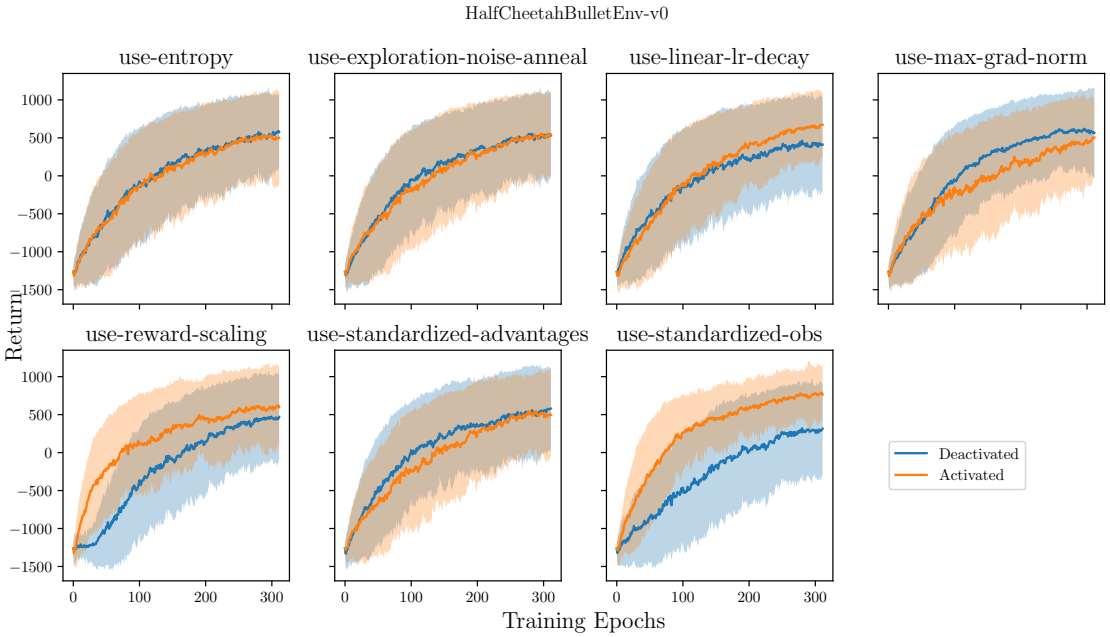

Figure 9: Impact of code-level ingredients on HalfCheetah task.

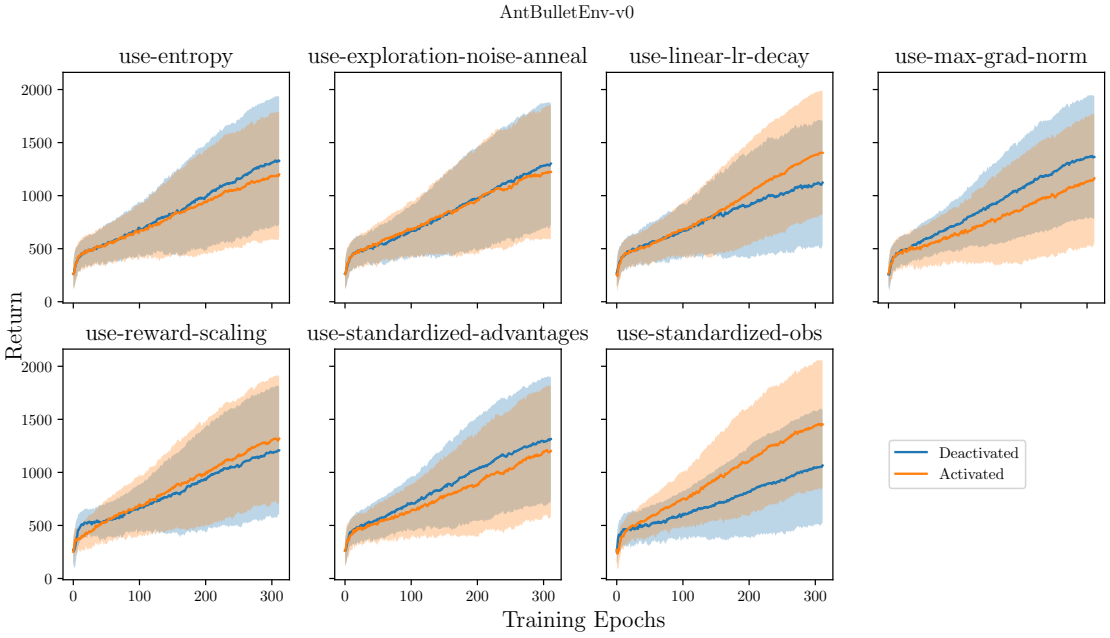

Figure 10: Impact of code-level ingredients on Ant task.

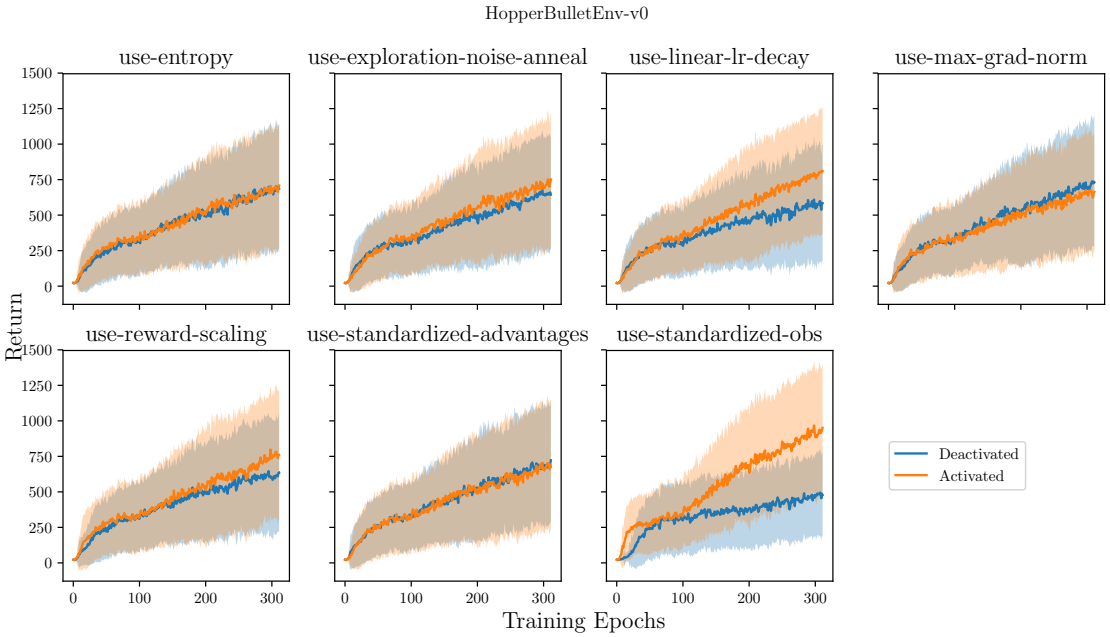

Figure 11: Impact of code-level ingredients on Hopper task.

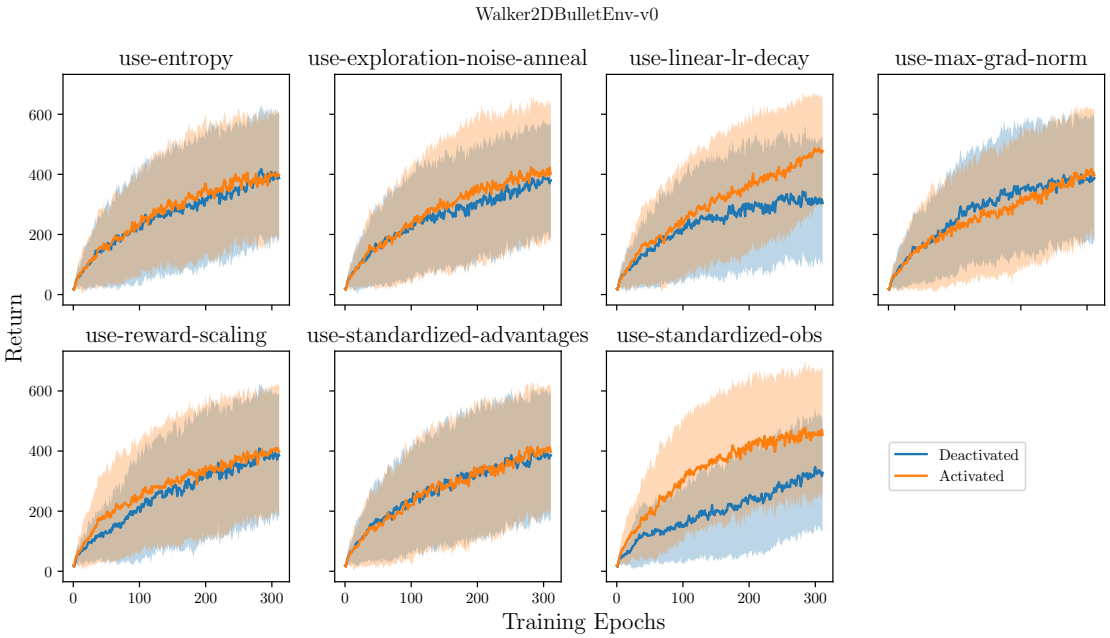

Figure 12: Impact of code-level ingredients on Walker2D task.

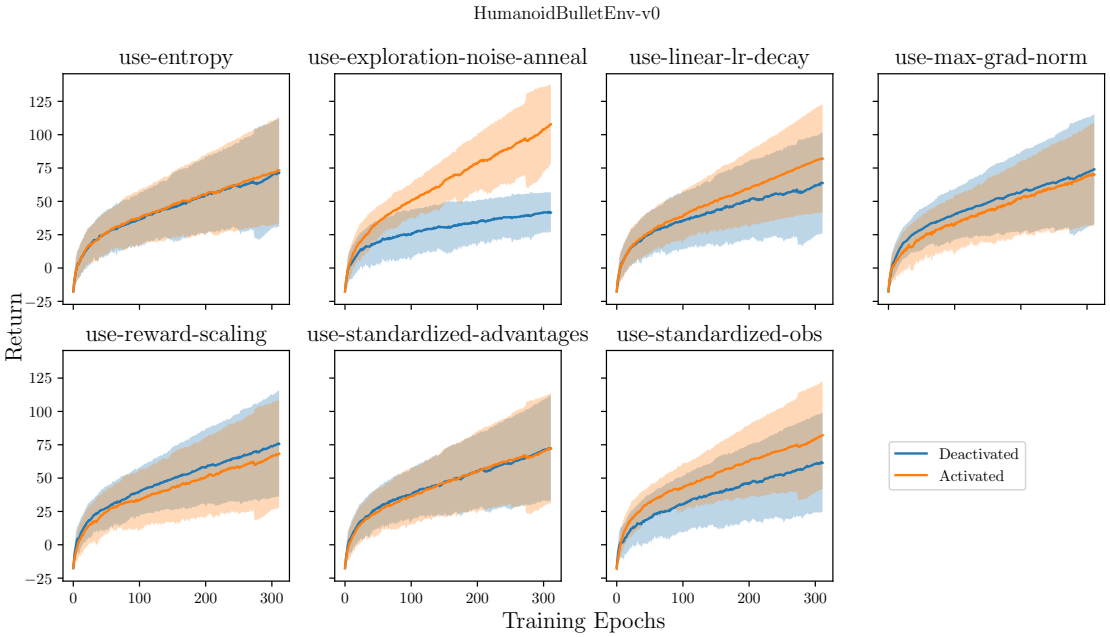

Figure 13: Impact of code-level ingredients on Humanoid task.

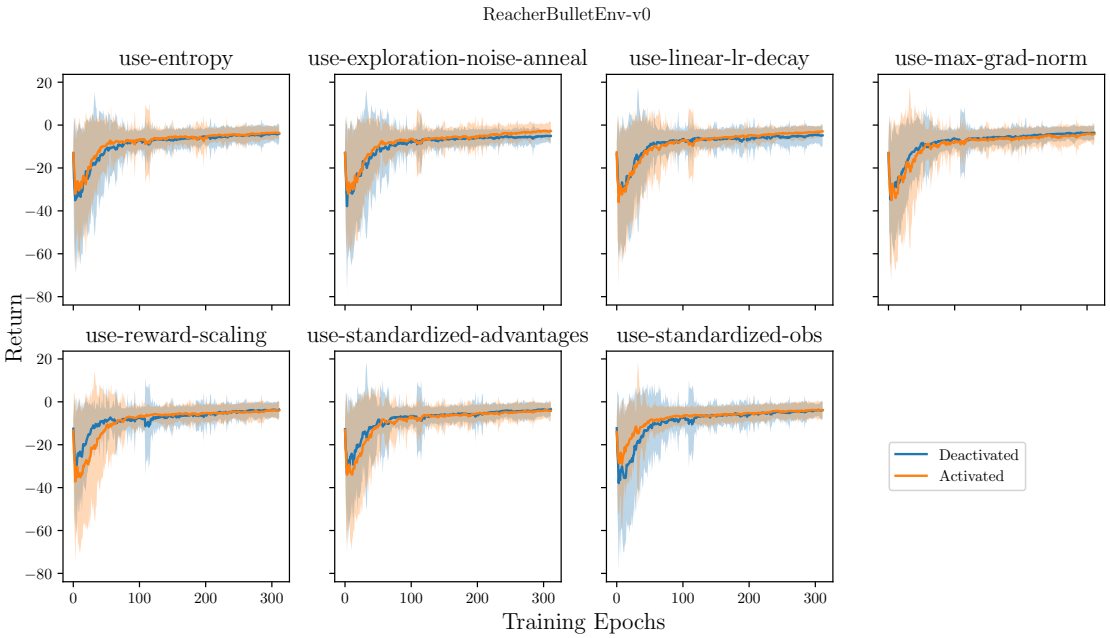

Figure 14: Impact of code-level ingredients on Reacher task.

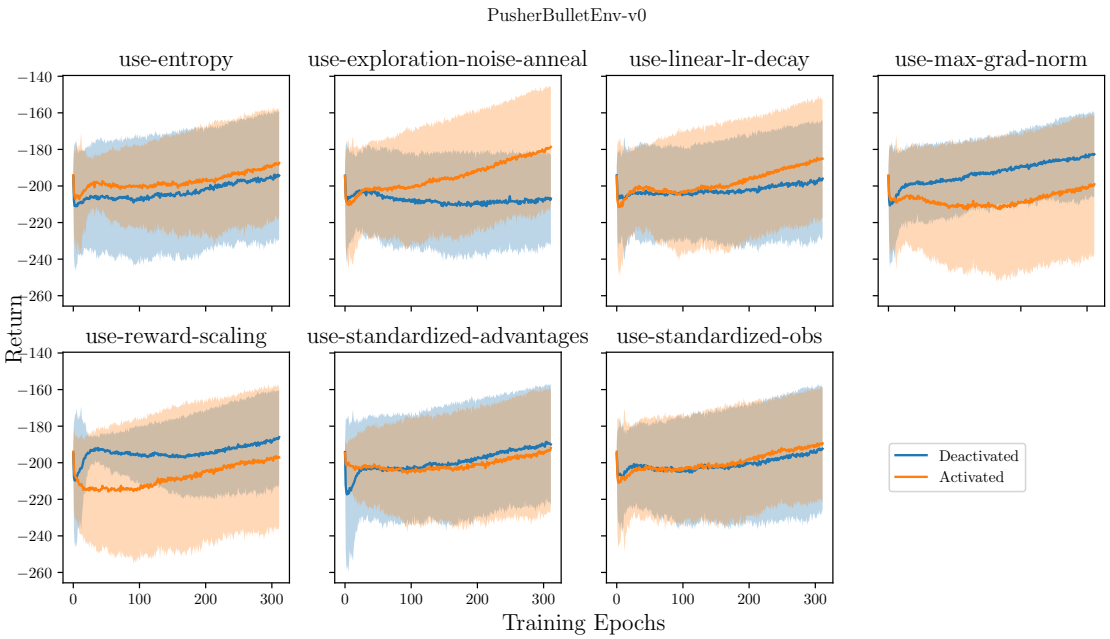

Figure 15: Impact of code-level ingredients on Pusher task. Note that less explorative policies generate better reward and, hence, the reduction in exploration is responsible for better performance. Post-training evaluation showed no performance differences..

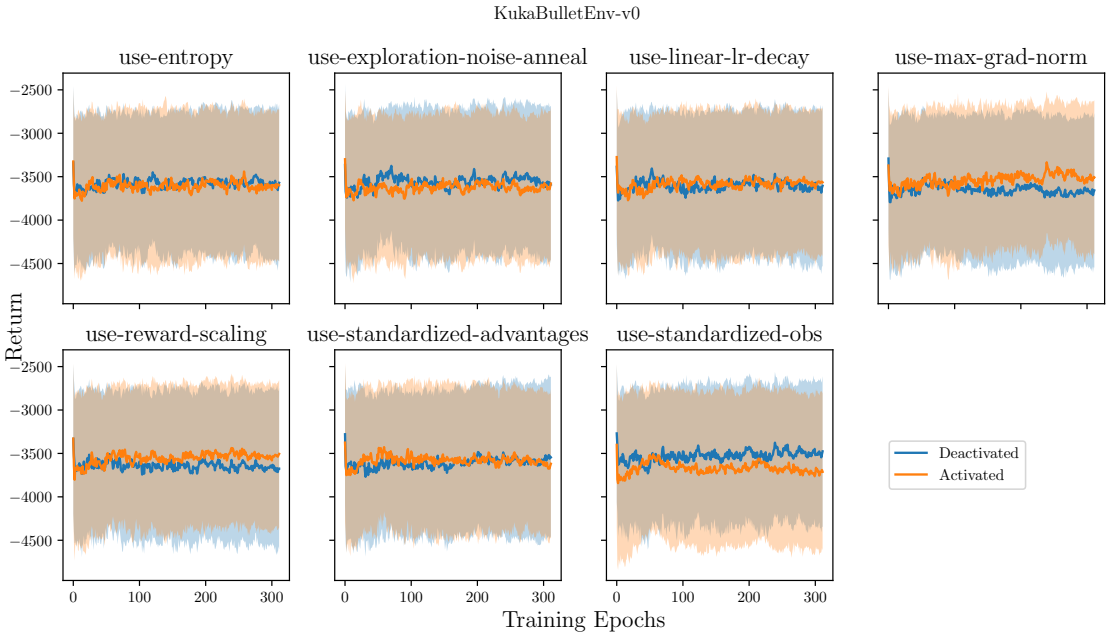

Figure 16: Impact of code-level ingredients on Kuka task. Note that we could not achieve performance gains with plain code-level ingredients. Only by the use of GAE or V-trace estimation methods, we could accomplish learning progress (see Experiments B).

Experiments B. Algorithmic Ingredients

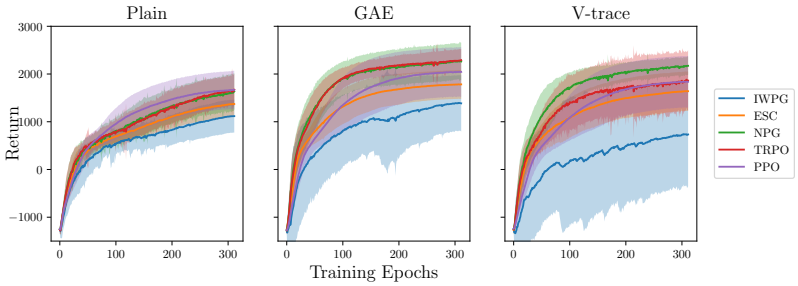

Figure 17: Impact of algorithmic ingredients on HalfCheetah task.

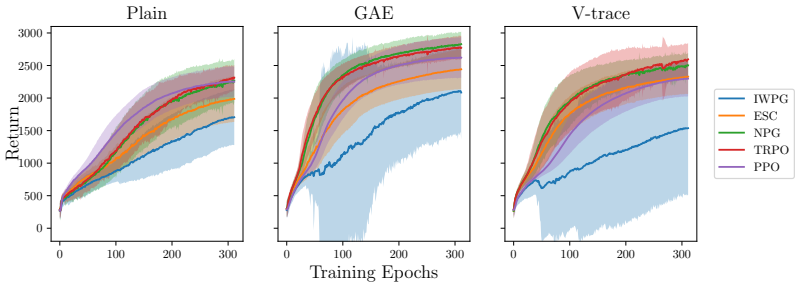

Figure 18: Impact of algorithmic ingredients on Ant task.

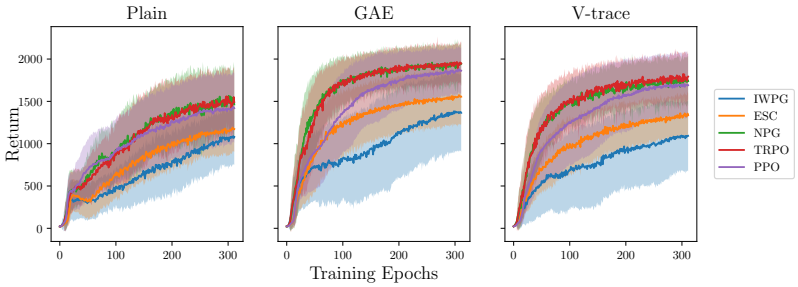

Figure 19: Impact of algorithmic ingredients on Hopper task.

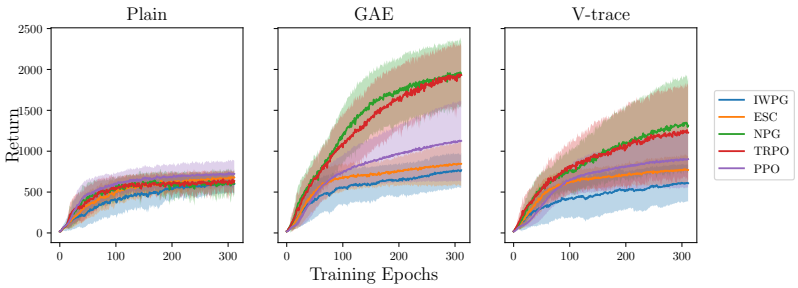

Figure 20: Impact of algorithmic ingredients on Walker task.

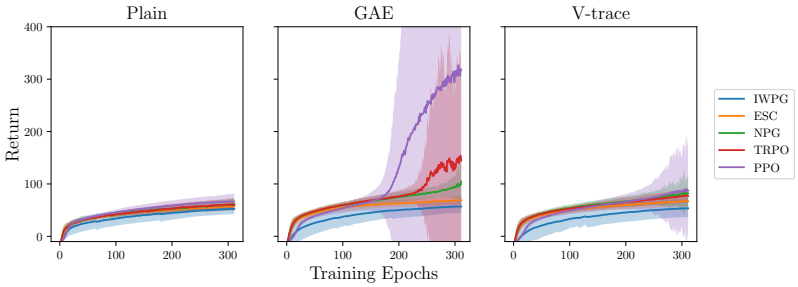

Figure 21: Impact of algorithmic ingredients on Humanoid task.

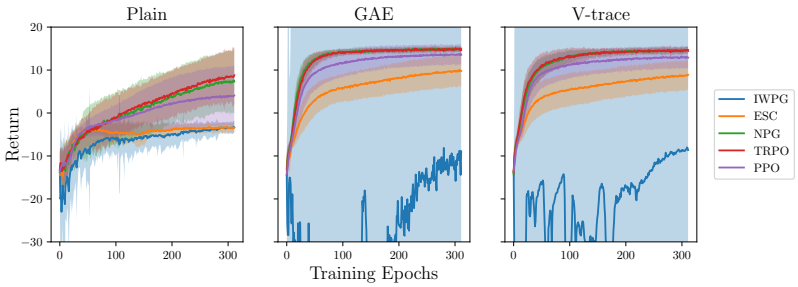

Figure 22: Impact of algorithmic ingredients on Reacher task.

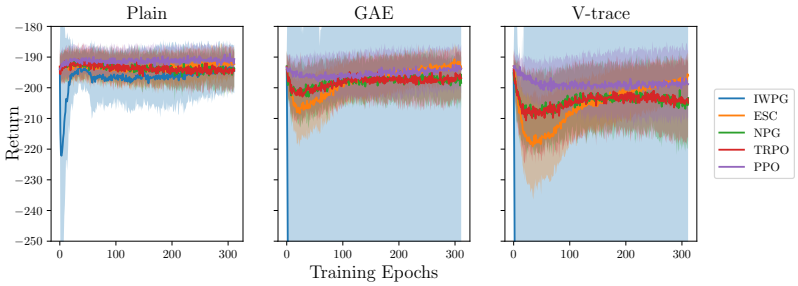

Figure 23: Impact of algorithmic ingredients on Pusher task.

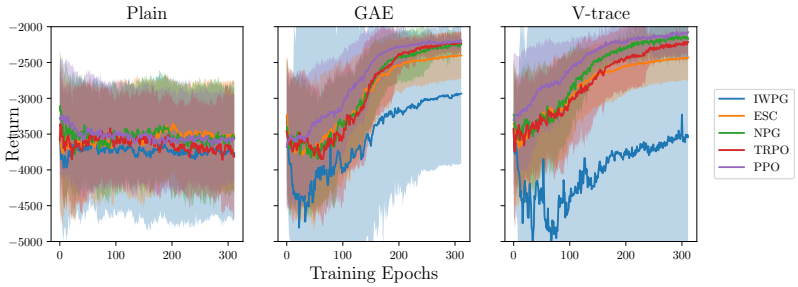

Figure 24: Impact of algorithmic ingredients on Kuka task.

Extended Analysis

We conducted a deeper analysis about the intricate interplay between specific ingredients. As experimental setup, we used the following ingredients as default:

- Code-Level Ingredients: observation standardization, linear learning rate annealing, reward scaling (only for locomotion tasks)
- Algorithmic ingredients: IWPG with GAE
- Structural Ingredients: Kaiming Initialization, Adam (LR=1.0e-4), no parameter sharing, Adam Eps = 1.0e-8

We used as number of training iterations $M = 80$ for the policy network.

Interplay between Variance Reduction and the LR Annealing

Figures 25 - 27 compare the impact of the learning rate annealing and the advantage reduction method in the three tasks: Ant, Walker, and Pusher. Each plot line is averaged over 32 independent random seeds from a grid search where learning rates $\alpha_\pi \in [1e-4, 2.5e-4, 5.0e-4, 1e-3]$ and numbers of policy training iterations $M \in [10, 20, 40, 80]$ were evaluated for each configuration two times. The decay of the learning rate resulted in decreasing step sizes in the policy parameter space over the training. This suggests that the learning rate annealing promotes the convergence to better local solutions.

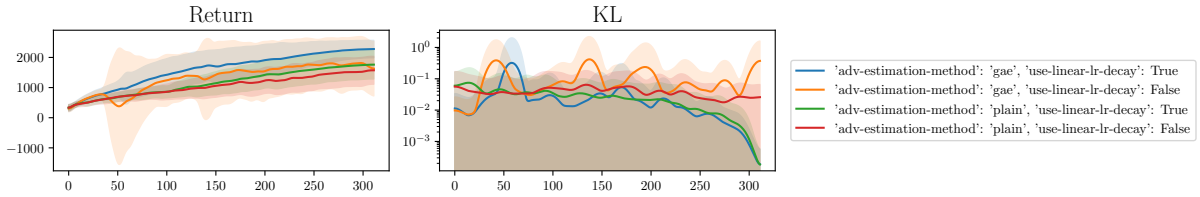

Figure 25: Ant Task.

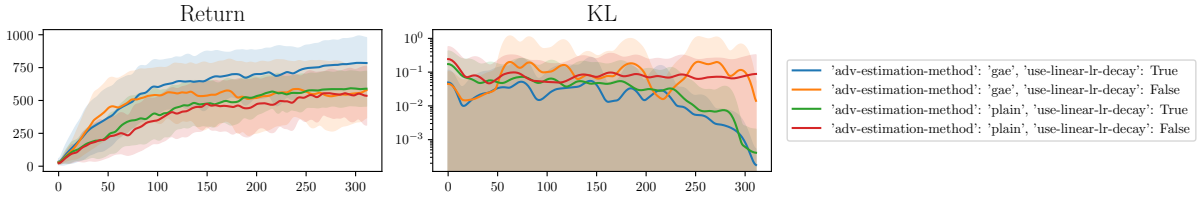

Figure 26: Walker Task.

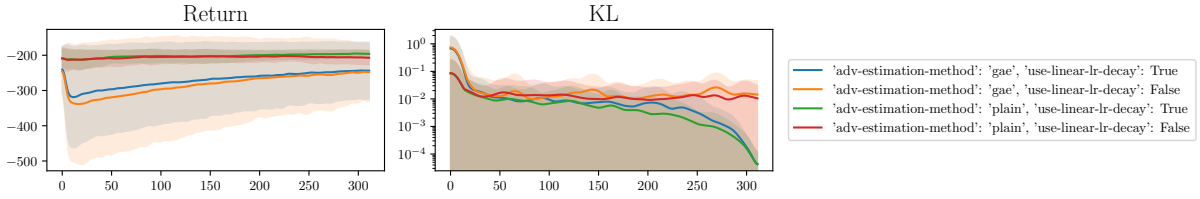

Figure 27: Pusher Task.

Interplay between Reward Scaling and Advantage Standardization

Here, we study the intricate interplay between Reward Scaling and Advantage Standardization. The results from Table 10 show that the best results over all eight tasks yields the combination when reward scaling is applied but no advantage standardization. Further, advantage standardization yields profits when no reward scaling is applied, but reduces performance if both are applied. We recommend to exclusively apply the one or the other, while reward scaling being the preferable one. The learning curves are depicted in Figure 28. Each curve is average over eight random seeds. We rank each combination based the final performance after 10^6 training steps when exploration noise is deactivated.

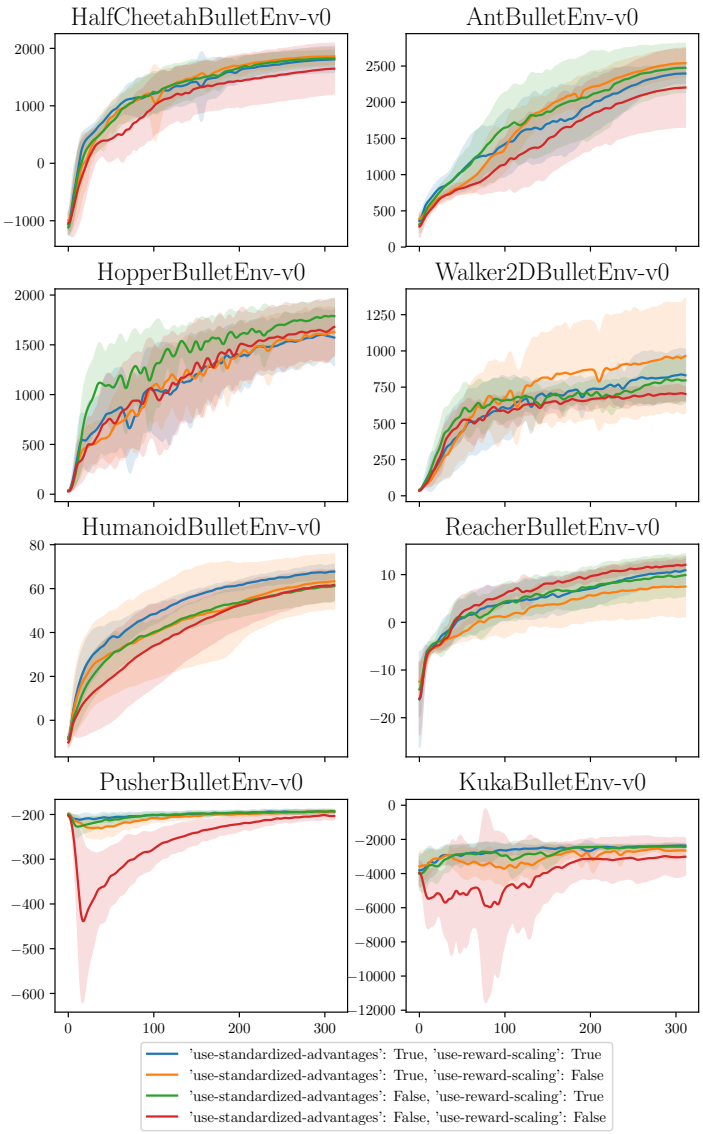

Figure 28: Interplay between Reward Scaling and Advantage Standardization. Curves represent the mean over eight seeds, while the shaded area shows the standard deviation.

Ranks	HalfCheetah	Ant	Hopper	Walker2D	Humanoid	Reacher	Pusher	Kuka	Mean Rank
Adv. St.: True, Rew. Sc.: True	2	3	4	2	1	2	3	1	2.25
Adv. St.: True, Rew. Sc.: False	3	1	3	1	3	4	1	3	2.38
Adv. St.: False, Rew. Sc.: True	1	2	1	3	2	3	2	2	2.00
Adv. St.: False, Rew. Sc.: False	4	4	2	4	4	1	4	4	3.38

Table 10: Rankings of the usage of Reward Scaling and Advantage Standardization.

Interplay between the Scale of Reward and the Usage of Reward Scaling

In this analysis, we tested the influence of the reward scale towards the learned policy performance. We multiplied the observed rewards from the environment by the factors $\nu \in [0.01, 0.1, 1, 10, 100]$ and investigated two advantage estimation methods: plain and GAE. As shown in Figure 29, applying reward scaling as code-level ingredients improved the performance when the rewards given by the environment are scaled by the values $\nu \in [0.01, 100]$, but less for reward scales $\nu \in [0.1, 1, 10]$. Overall, we did not recognize differences when using the factor $\nu = 0.1$. This suggests that reward scaling can be neglected when the environment is designed to exhibit rewards that are already well-scaled in magnitudes.

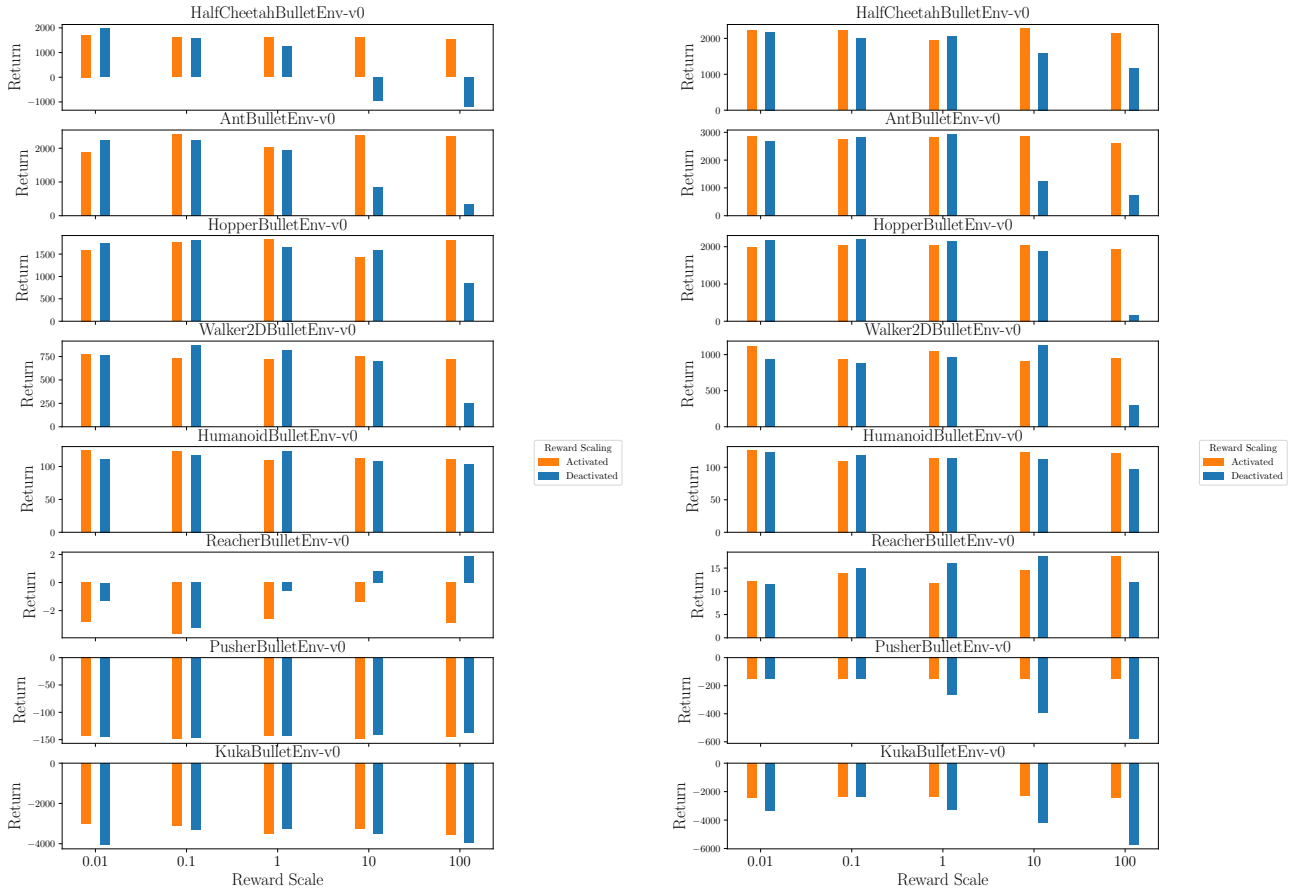

Figure 29: Influence of reward scale on policy performance. (Left) Plain advantage estimation is used. (Right) GAE as advantage estimation method is applied.