Phasic Policy Gradient (PPG)

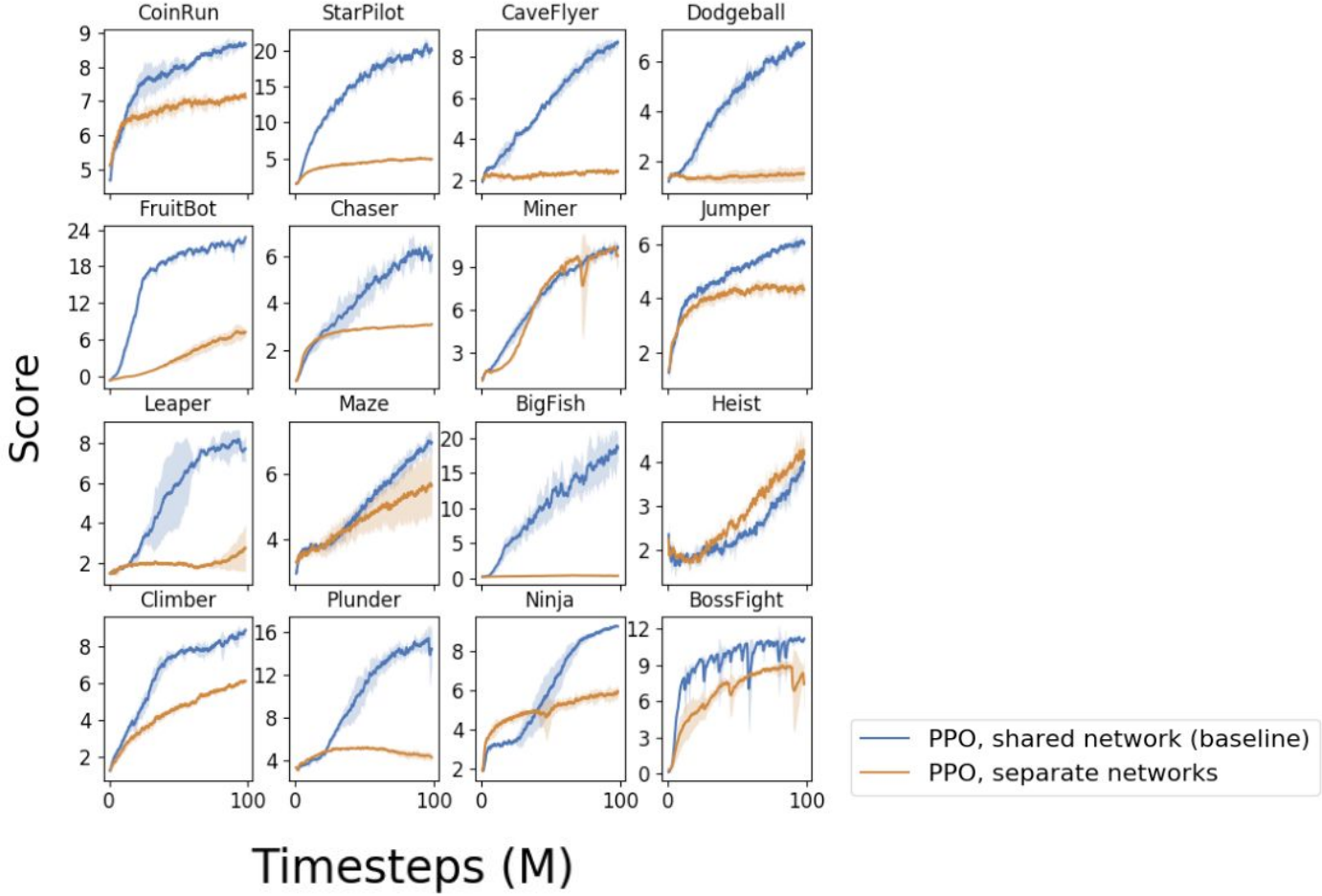
Karl Cobbe, Jacob Hilton, Oleg Klimov, John Schulman

OpenAI

Introduction and Motivation

- PPG separates policy and value function training into distinct phases
- To share or not to share features between policy and value networks?
    - Pro: valuable information is shared between both networks
    - Con: non-trivial interference between objectives
- PPG aims to achieve the best of both worlds

Importance of policy and value function feature sharing



Experiments performed on Procgen Benchmark

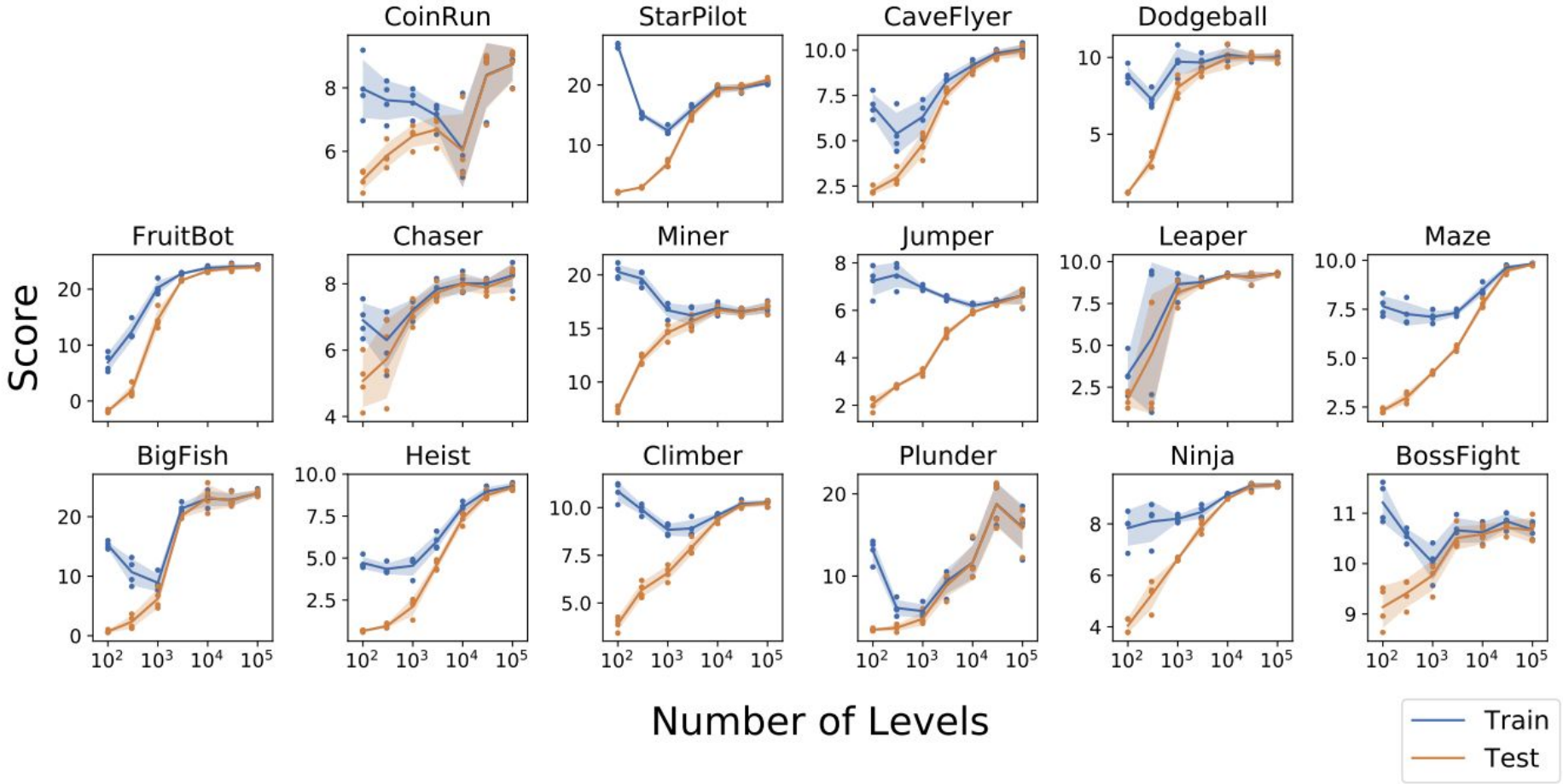
Why use Procgen Benchmark?

- Many common RL benchmarks ignore generalization
- Do agents learn robust skills or memorize trajectories?
- Atari: high diversity across envs, but low diversity within a single env

Design Principles

- High diversity
- Mimic the style of Atari (and Gym Retro) games
- Fast evaluation (1000s steps/sec on single CPU core)
- High level solvability (>99%)
- Shared action space: 15 dimensional discrete
- Shared observation space: 64x64x3 RGB

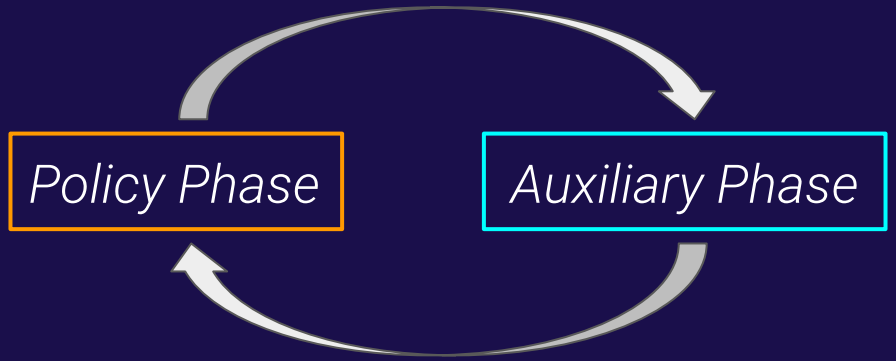
Generalization from Finite-Level Training Sets



The Downside of Shared Networks

- The policy and value function must be trained on the same data (i.e. the same level of sample reuse)
- There will be some amount of interference between policy and value function objectives
- With PPG, we can:
    - Reduce interference between policy and value function optimization
    - More aggressively train of the value function (using higher sample reuse)

Phasic Policy Gradient (PPG) Overview



for each of R=32 rollouts:

Perform standard PPO update

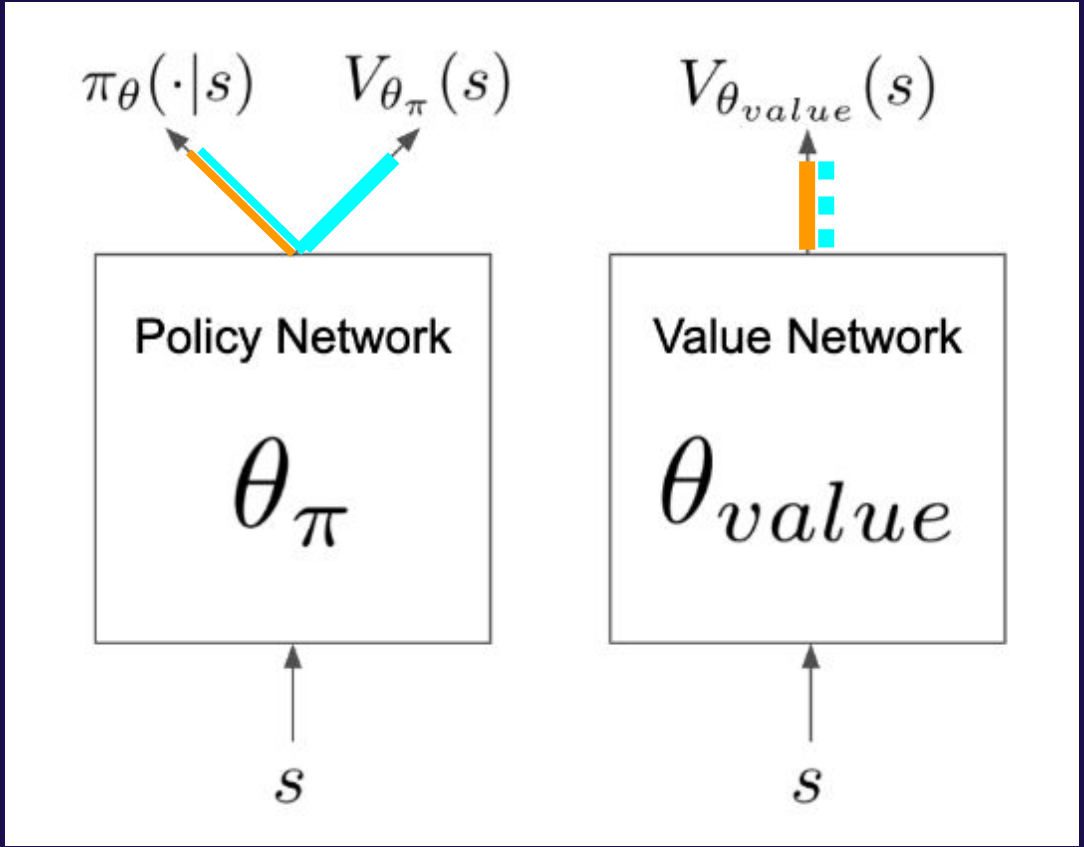
Store all states and value targets in replay buffer

for each of E=6 epochs over replay buffer:

Optimize value function while cloning prior policy

Policy Phase

Auxiliary Phase



Phasic Policy Gradient (PPG) Losses

Policy Phase:

$$L^{clip} = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \qquad r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

$$L^{value} = \hat{\mathbb{E}}_t \left[\frac{1}{2}(V_{\theta_V}(s_t) - \hat{V}_t^{\text{targ}})^2 \right]$$

Auxiliary Phase:

$$L^{joint} = L^{aux} + \beta_{clone} \cdot \hat{\mathbb{E}}_t \left[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]$$

$$L^{aux} = \frac{1}{2} \cdot \hat{\mathbb{E}}_t \left[(V_{\theta_\pi}(s_t) - \hat{V}_t^{\text{targ}})^2 \right]$$

Phasic Policy Gradient (PPG) Pseudocode

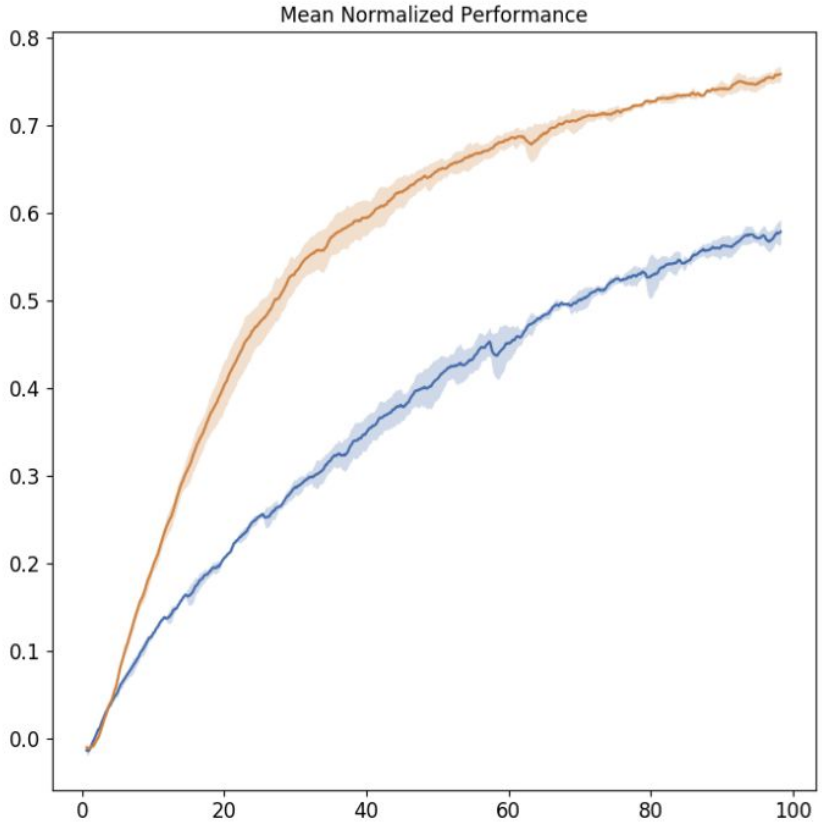
for phase $= 1, 2, \ldots$ **do**
  Initialize empty buffer B
  for iteration $= 1, 2, \ldots, N_\pi$ **do** ▷ Policy Phase
    Perform rollouts under current policy π
    Compute value function target $\hat{V}_t^{\text{targ}}$ for each state s_t
    for epoch $= 1, 2, \ldots, E_\pi$ **do** ▷ Policy Epochs
      Optimize $L^{clip} + \beta_S S[\pi]$ wrt θ_π
    for epoch $= 1, 2, \ldots, E_V$ **do** ▷ Value Epochs
      Optimize L^{value} wrt θ_V
    Add all $(s_t, \hat{V}_t^{\text{targ}})$ to B
  Compute and store current policy $\pi_{\theta_{old}}(\cdot|s_t)$ for all states s_t in B
  for epoch $= 1, 2, \ldots, E_{aux}$ **do** ▷ Auxiliary Phase
    Optimize L^{joint} wrt θ_π, on all data in B
    Optimize L^{value} wrt θ_V, on all data in B

Hyperparameters

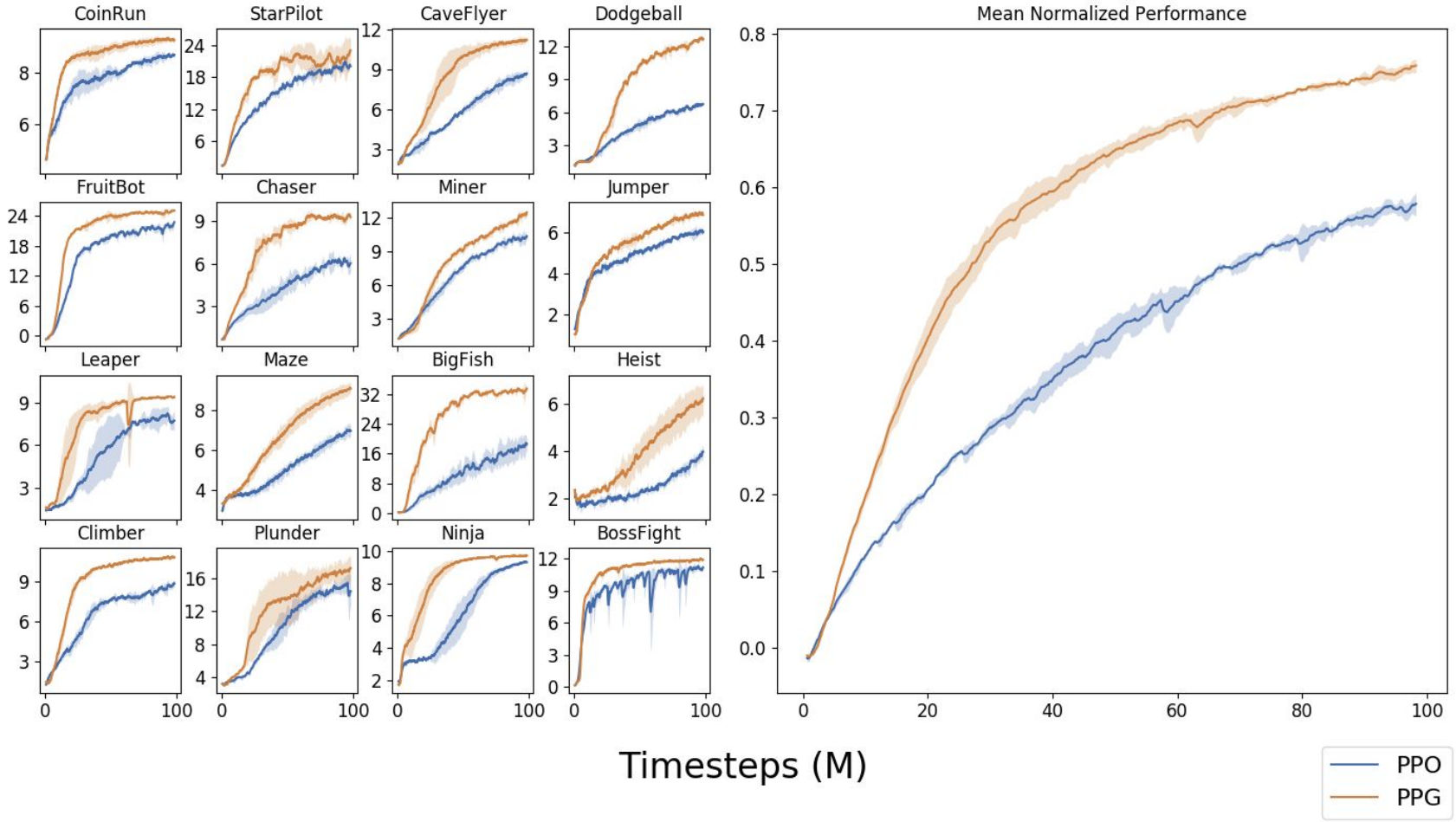
- N_{π} controls the length of each policy phase.
- E_{π} and E_{V} control the (policy phase) sample reuse for the policy and value function respectively
- Note: E_{V} influences the training of the true value function, not the auxiliary value function.
- E_{aux} controls the number of auxiliary epochs performed across all data in the replay buffer.
- It is usually by increasing E_{aux}, rather than E_{V}, that we increase sample reuse for value function training.
- β_{clone} controls the relative weight of the auxiliary objective and the policy stabilization objective.

PPO vs PPG

- Performance improvement is consistent
- PPO baseline is well tuned
- Policy sample reuse is similar to baseline
- Value function sample reuse is ~2x baseline
- We need careful ablations to understand the source of the improvement

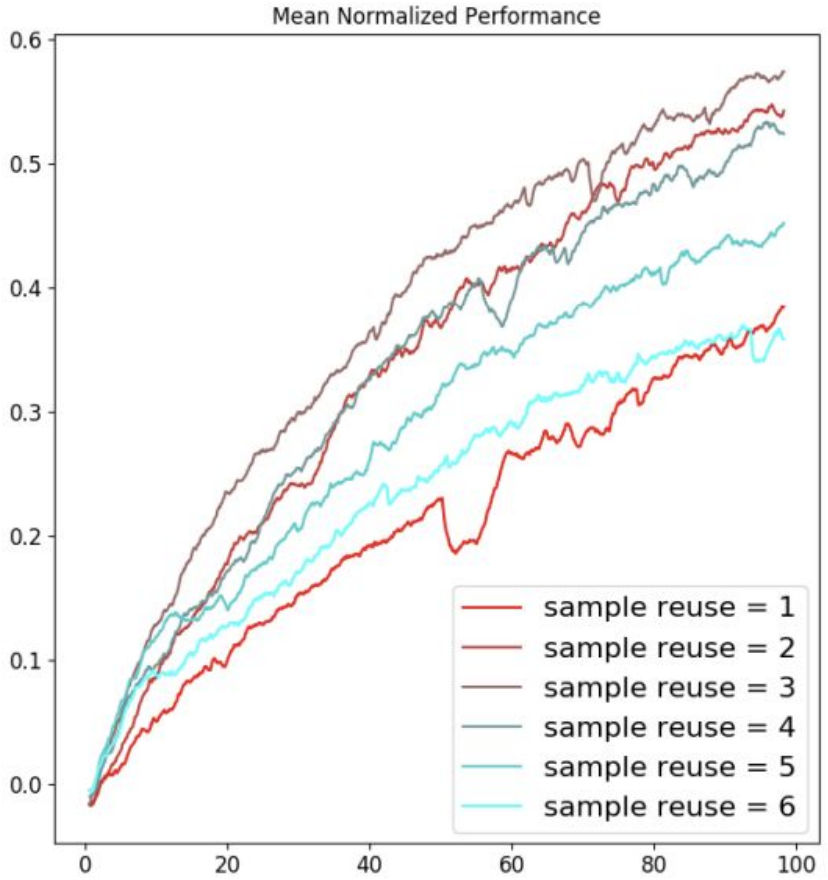


Comparison to PPO

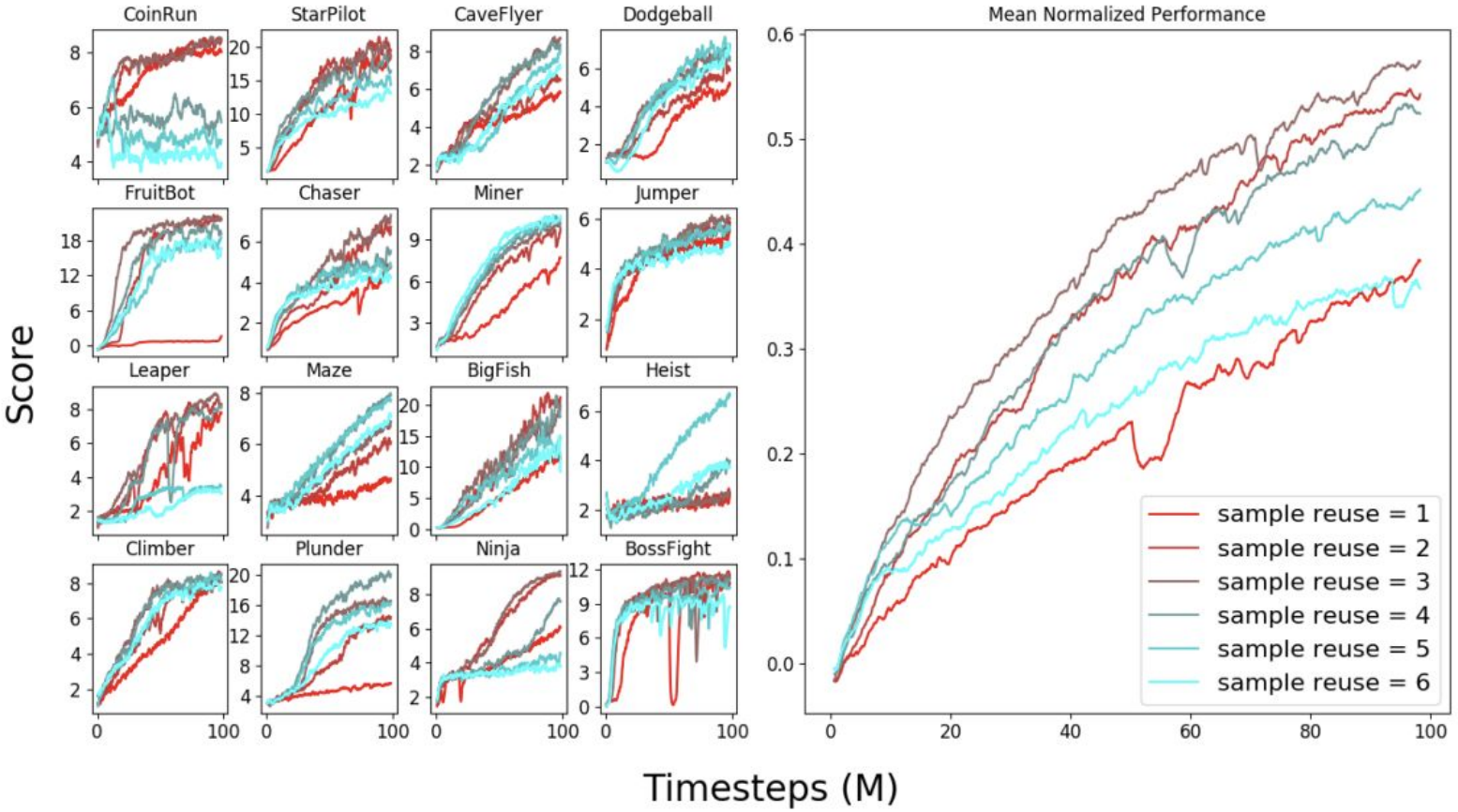


PPO Sample Reuse (policy and value function)

- Separate networks: can vary sample reuse for pi/vf, but performance is very poor.
- PPG allows us to vary pi/vf sample reuse while sharing features.
- Anecdotally, sharing representations is more important in complex visual environments
- For PPO, sample reuse = 3 is empirically optimal

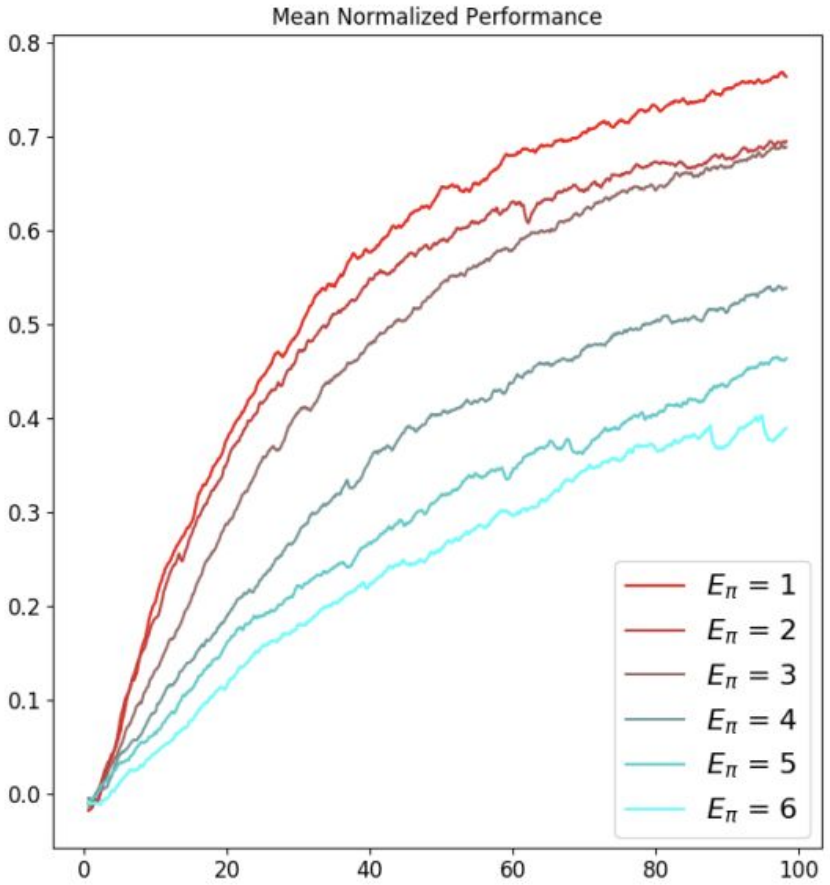


For PPO, sample reuse = 3 is empirically optimal

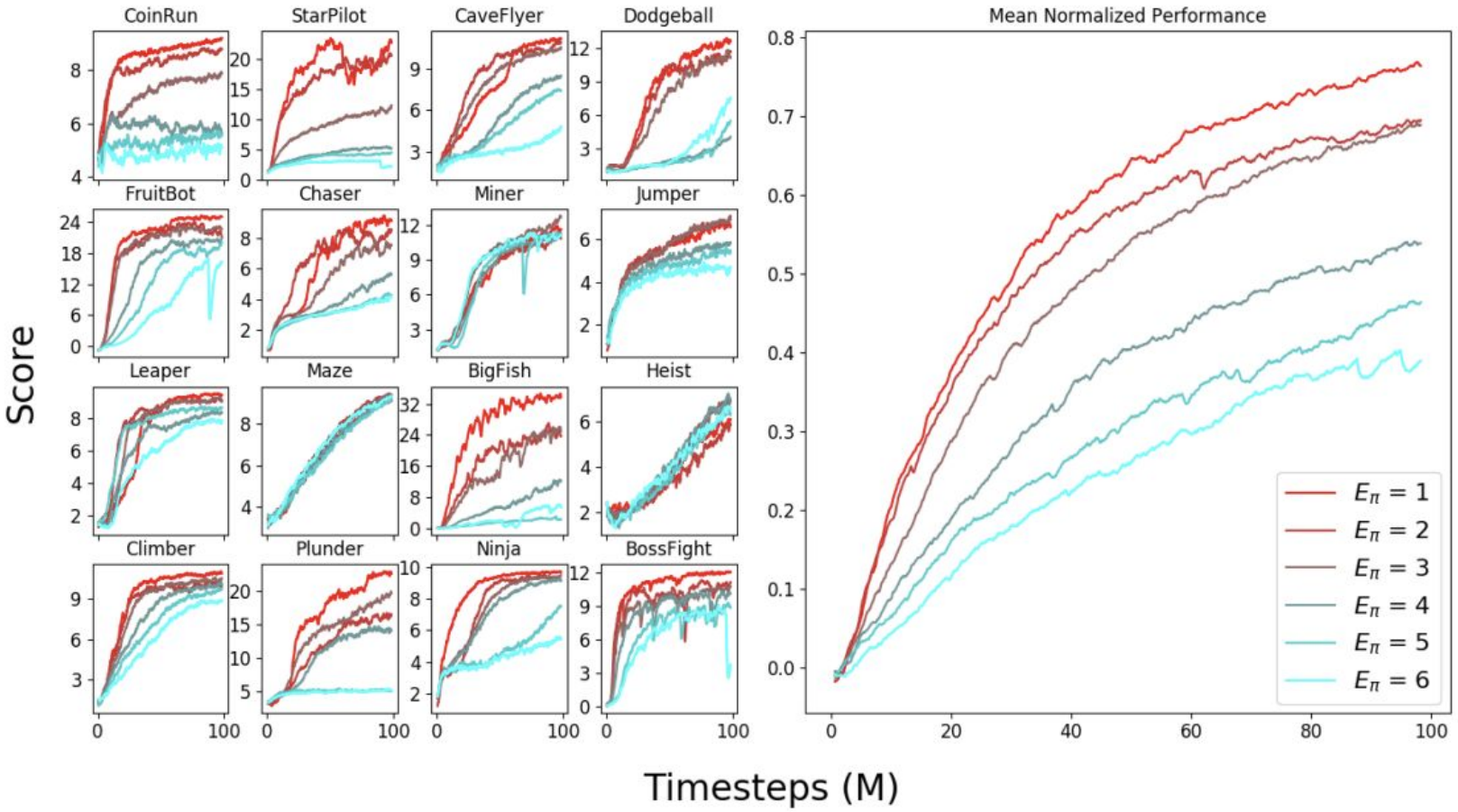


Policy Sample Reuse

- Higher policy sample reuse isn't beneficial
- So why does PPO benefit from higher sample reuse?
- VF sample reuse has two possible benefits:
  - Reduce policy gradient variance
  - Train better representations
- Why are additional policy epochs harmful in PPG?
  - Overfitting?
  - Destructively large updates?



Higher policy sample reuse isn't beneficial



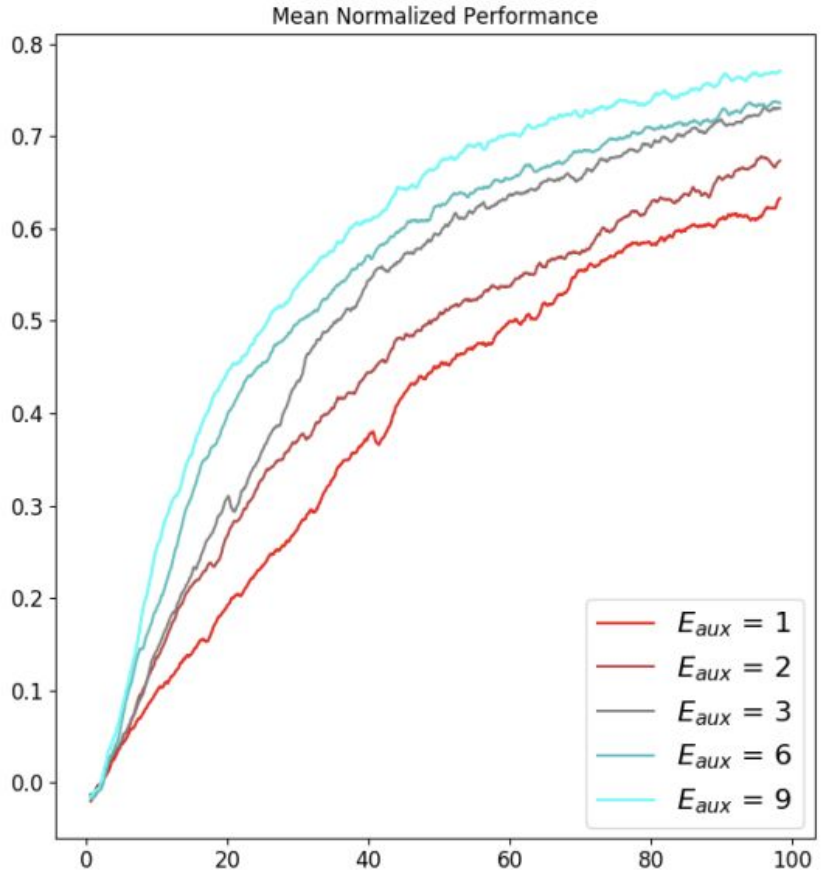
Value Function Sample Reuse

- Controlled by # epochs / auxiliary phase (E_{aux})
- Higher E_{aux} runs the risk of overfitting to recent data
- Lower E_{aux} can lead to slower training
- Each loss, L^{joint} and L^{value}, offers different improvement

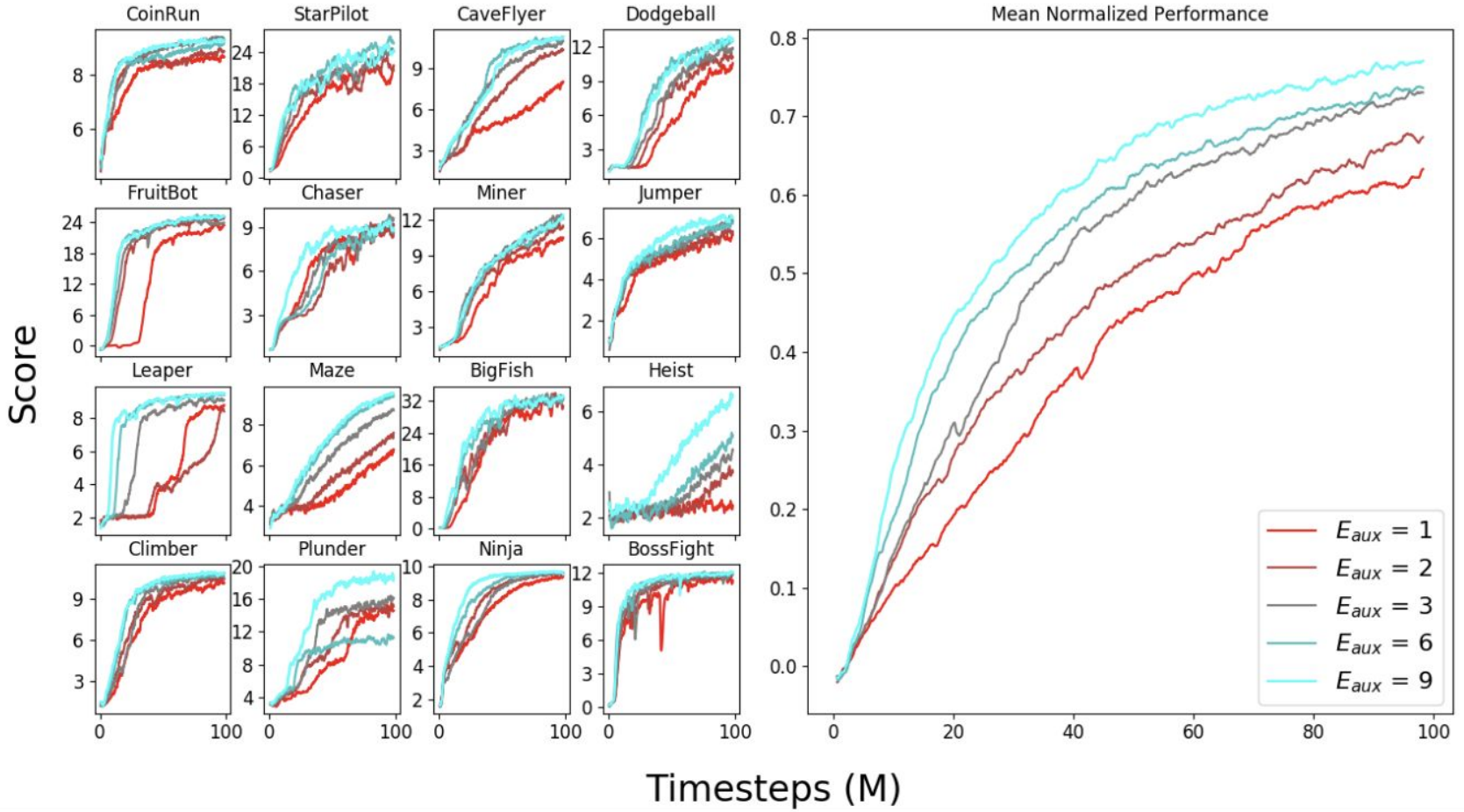
$$L^{joint} = L^{aux} + \beta_{clone} \cdot \hat{\mathbb{E}}_t \left[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]$$

$$L^{value} = \hat{\mathbb{E}}_t \left[\frac{1}{2} (V_{\theta_V}(s_t) - \hat{V}_t^{\text{targ}})^2 \right] \quad L^{aux} = \frac{1}{2} \cdot \hat{\mathbb{E}}_t \left[(V_{\theta_\pi}(s_t) - \hat{V}_t^{\text{targ}})^2 \right]$$

- Note: we don't rebootstrap value targets
- Impact of L^{joint} and L^{value} likely to vary by environment



Higher value function sample reuse improves sample efficiency

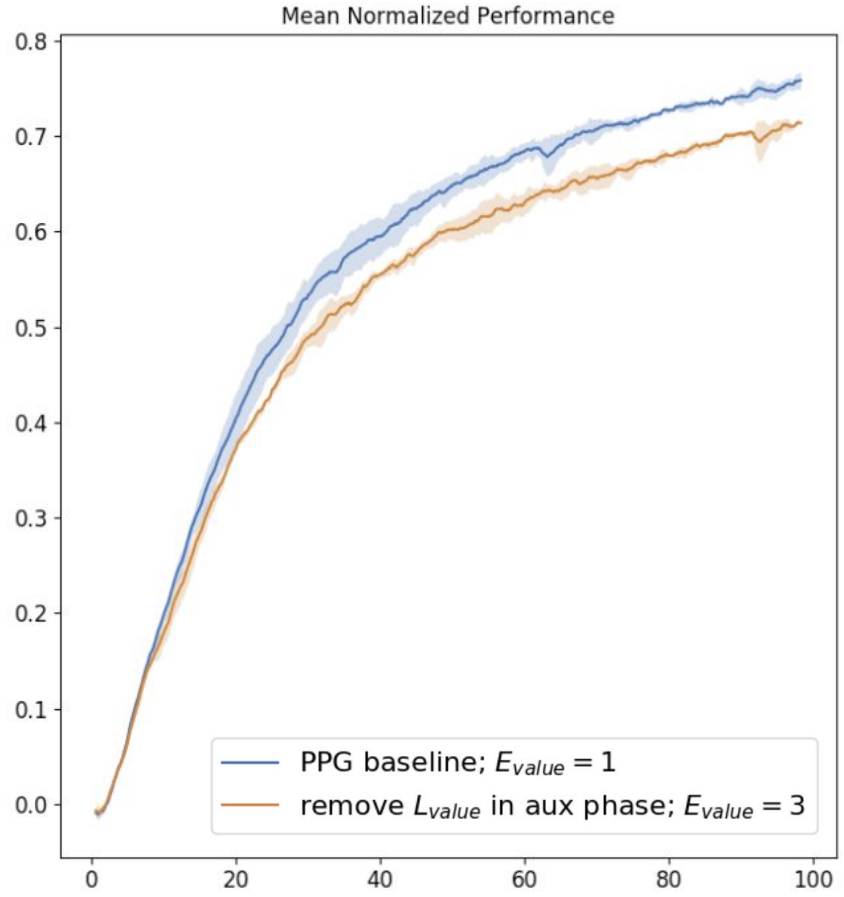


Comparing L^{value} and L^{joint}

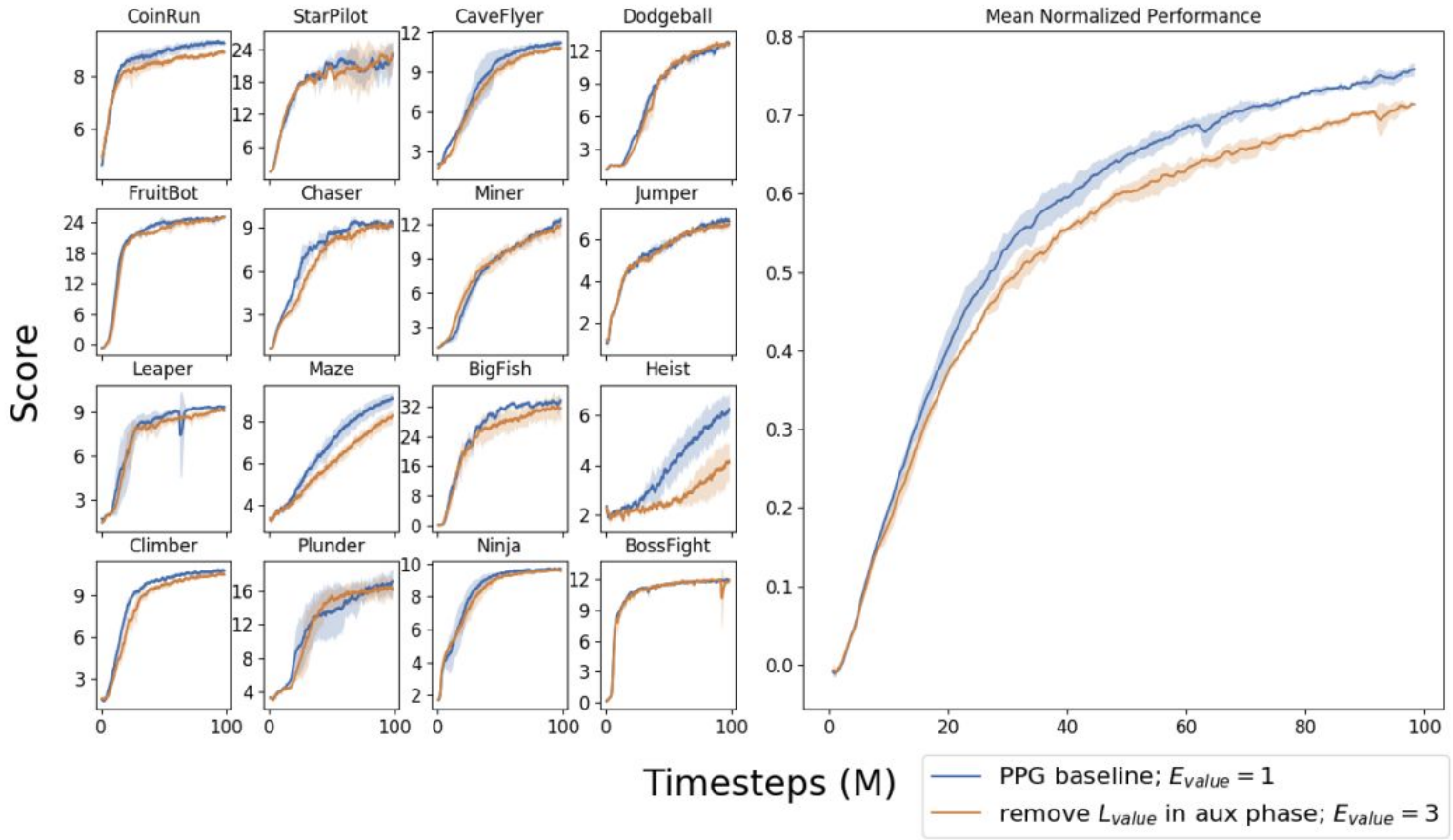
- We can remove L^{value} from the auxiliary phase
- After re-adjusting E_{value}, performance barely suffers

$$L^{joint} = L^{aux} + \beta_{clone} \cdot \hat{\mathbb{E}}_t \left[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]$$

$$L^{value} = \hat{\mathbb{E}}_t \left[\frac{1}{2} (V_{\theta_V}(s_t) - \hat{V}_t^{\text{targ}})^2 \right] \quad L^{aux} = \frac{1}{2} \cdot \hat{\mathbb{E}}_t \left[(V_{\theta_\pi}(s_t) - \hat{V}_t^{\text{targ}})^2 \right]$$

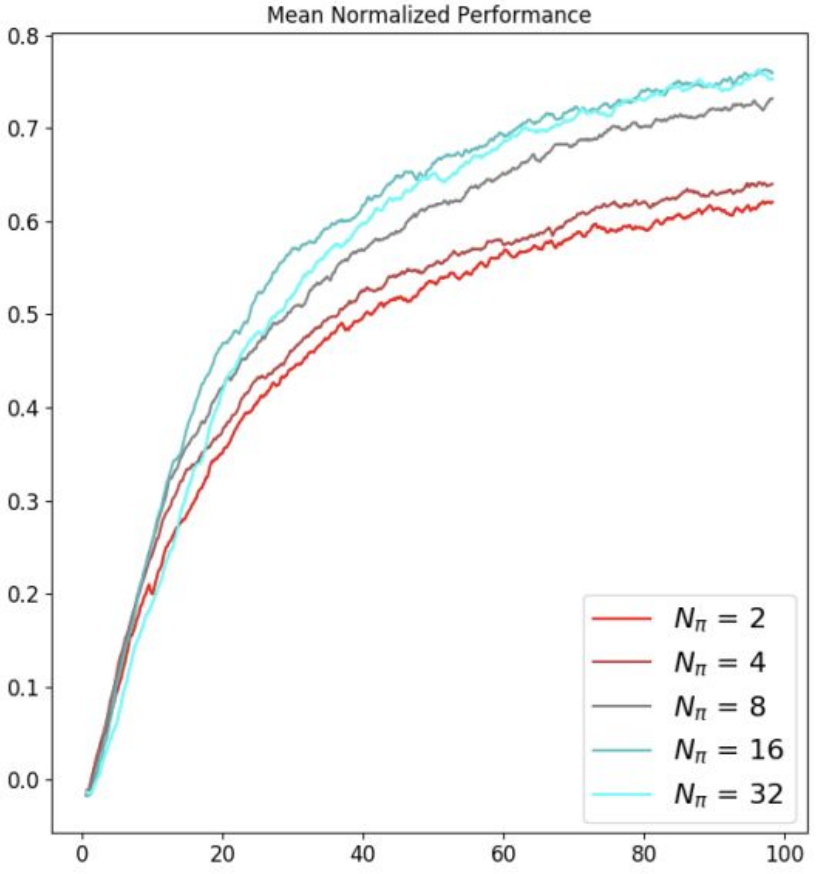


Comparing L^{value} and L^{joint}

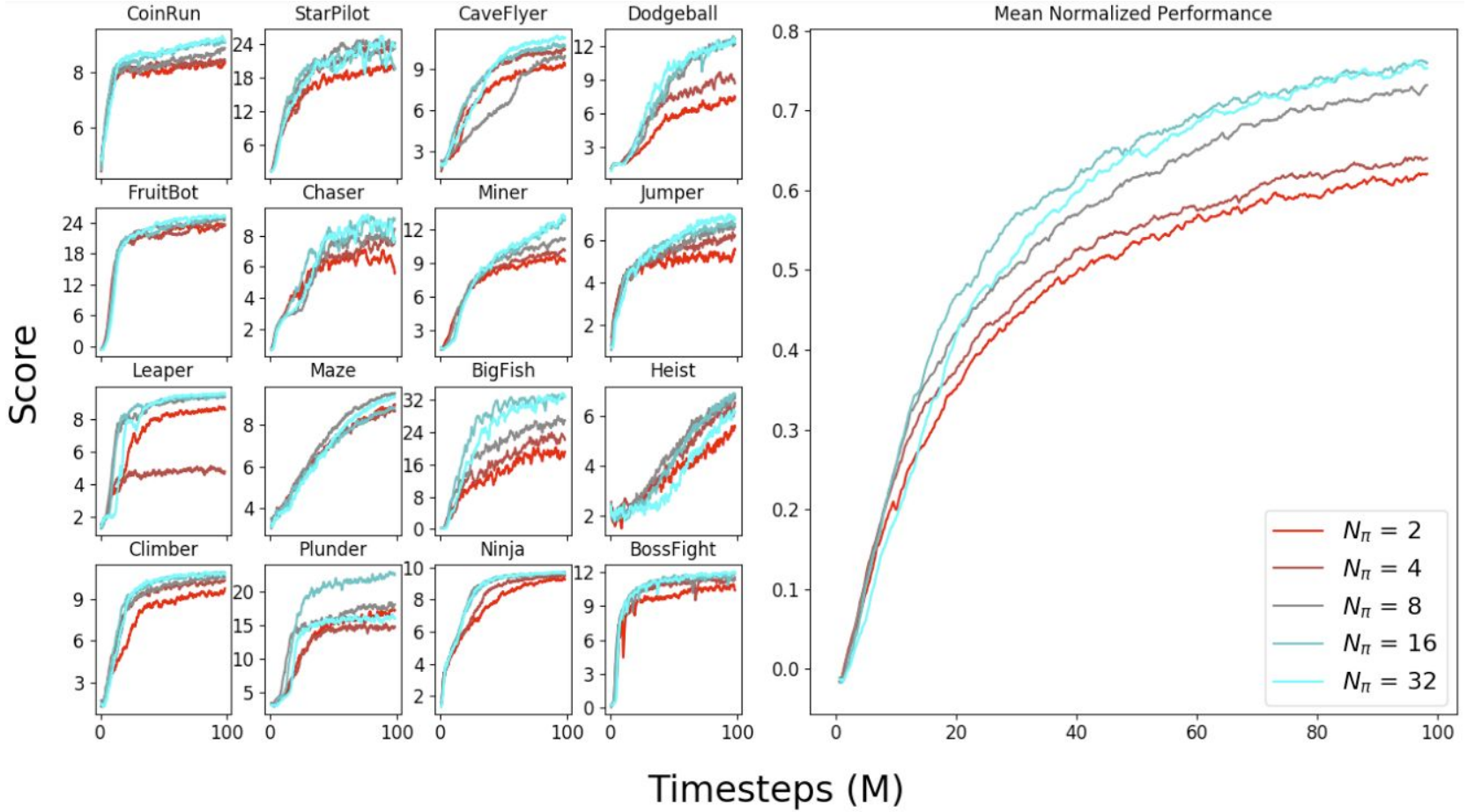


Auxiliary Phase Frequency

- Our earlier iterations alternated between policy and value function training after every rollout
- Each auxiliary phase causes to some interference
- Infrequent aux phases: stale representations
- Frequent aux phases: too much interference
- Policy sensitivity likely to vary by environment
- Ideally we would unify phases



Increasing auxiliary phase frequency degrades performance

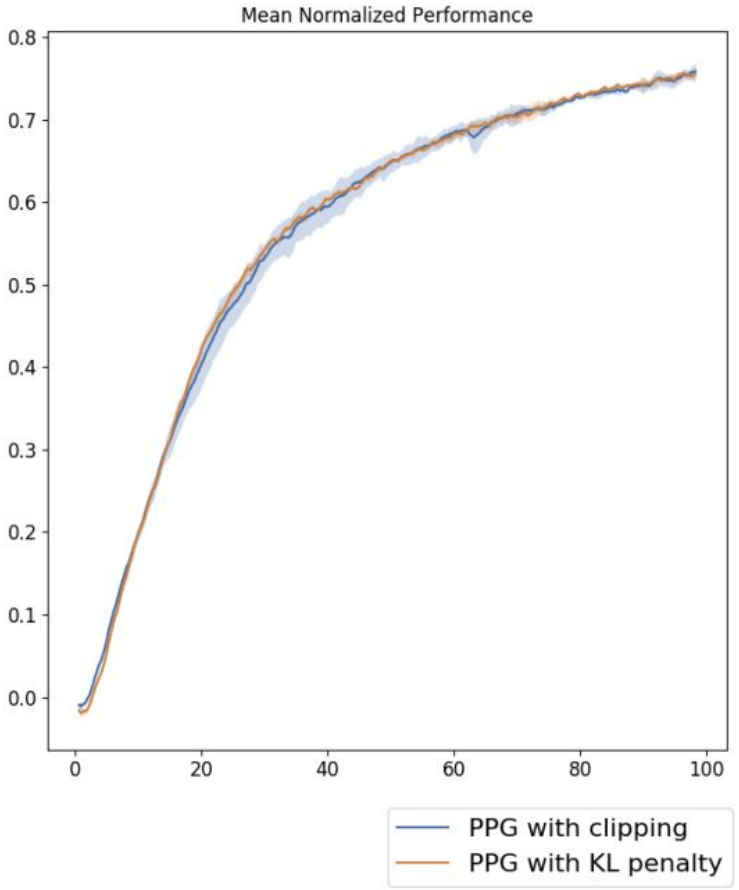


KL Penalty vs Clipping

- PPO originally also proposed an adaptive KL penalty
- KL penalty and clipping are both designed to prevent destructive updates to the policy
- Both objectives penalize the policy for updating too quickly
- βπ controls the weight of the KL penalty
- In practice, the KL penalty performs similarly to clipping
- Future work may wish to build upon either objective

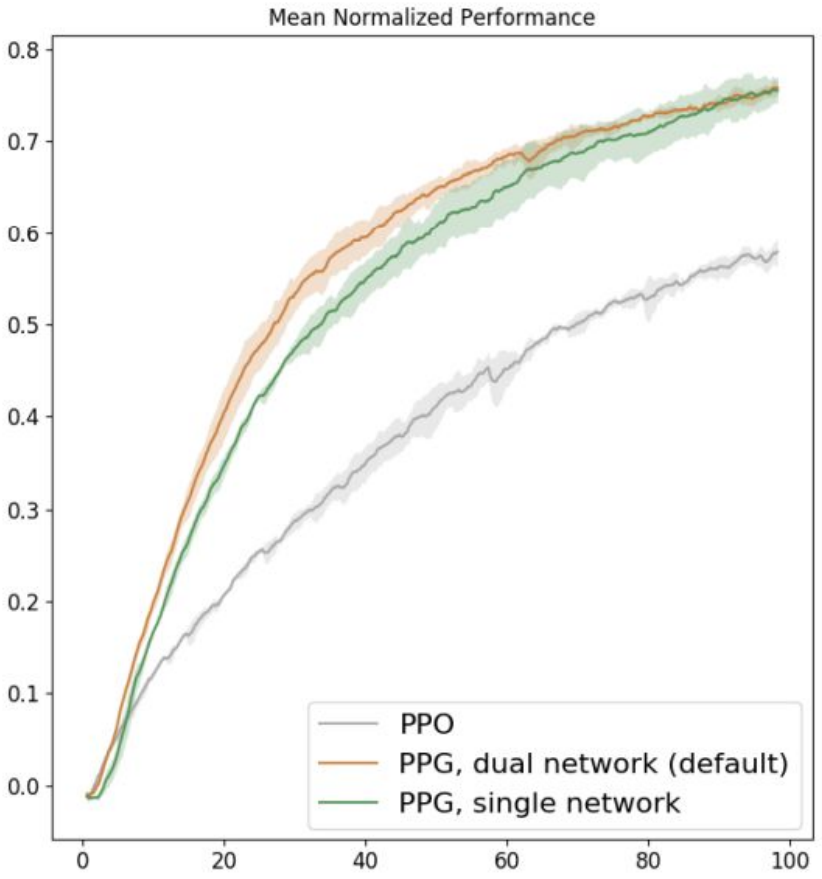
$$L^{clip} = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

$$L^{KL} = \hat{\mathbb{E}}_t \left[-\hat{A}_t \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} + \beta_\pi \cdot KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]$$

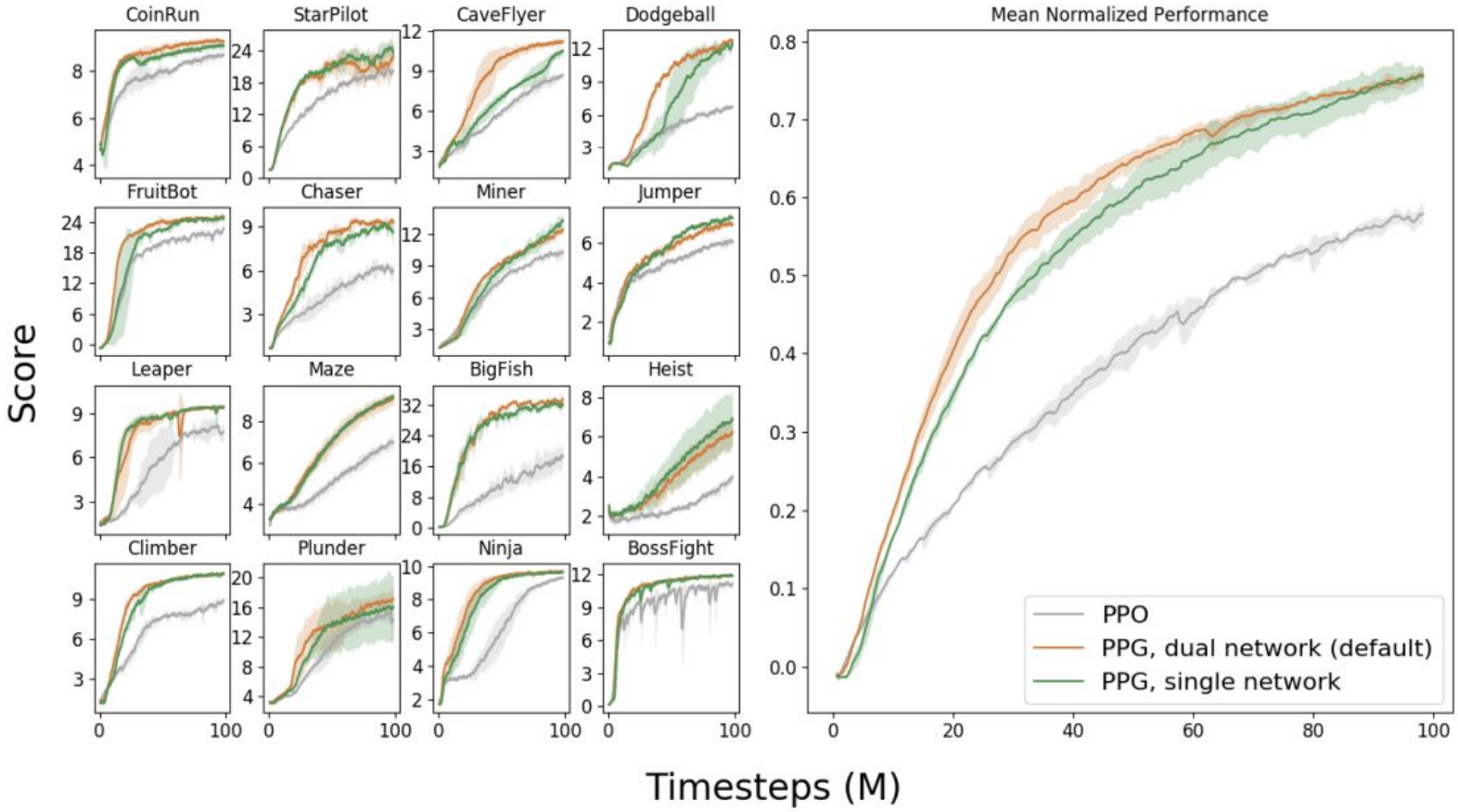


Single Network PPG

- By default, PPG uses 2x memory as PPO
- Single Network PPG halves memory footprint, with only slight drop in performance
- Key idea: detach value function gradient during policy phases
- Both variants of PPG:
  - Have no policy gradient interference
  - Benefit from sharings representations
- Single Network PPG uses less wall clock time



Single Network PPG



PPG Benefits

- Share features between policy and value function, while mitigating policy interference during training
- Independently vary sample reuse of the policy and the value function
- Maximally utilize the value function as an auxiliary objective

PPG Drawbacks

- Increase parameter count by a factor of 2 (can be avoided with Single Network PPG)
- Increased sample reuse leads to increased wall clock time

Thanks for listening!

- Special thanks to my co-authors John Schulman, Jacob Hilton and Oleg Klimov.
- Check out PPG code here: https://github.com/openai/phasic-policy-gradient