ABC-DRS: Domain-Randomized Speeding with Adaptive Beta Clipping for Meta-Learned Autonomous Racing

Roberto Ligeralde, Kevin Liu, Ethan Yu, Nikhil Kumar
University of Pennsylvania, Philadelphia, USA
{roblig22, kliu2360, ethanyu}@seas.upenn.edu, kumarnik@sas.upenn.edu

***Abstract*—Vehicle controllers trained with reinforcement learning have been described as achieving super-human racing performance, setting lap times comfortably faster than those of expert human drivers. However, these controllers are *super-human only for a particular vehicle on a particular track*—change either even slightly and a human driver will adapt while the policy fails to generalise. A truly super-human policy must therefore outperform experts *across* tracks and hardware platforms. We attack this problem with a meta-learning formulation that learns F1TENTH autonomous-racing policies capable of rapid adaptation to unseen track layouts, vehicle dynamics and opponent behaviours. Our method augments Meta-Q-Learning with Adaptive Beta Clipping (ABC), a self-tuning variance-reduction scheme that clips task-propensity weights using running batch statistics. Empirically, ABC tightens the critic-loss distribution, lowering its variance by $2.7\times$—thereby (potentially) enabling faster and more stable task adaptation. Please refer to our github for implementation details.**

***Index Terms*—Autonomous Racing, Meta-learning, Multi-Agent Reinforcement Learning**

I. Introduction

Recently, defending Formula 1 World Champion Max Verstappen tested out a Ferrari 296 GT3 on the Nürburgring circuit. Despite driving a radically different vehicle on a track which has not hosted F1 in several decades, Verstappen set an unofficial track record within a couple days. That is to say, human racers are capable of rapid adaptation to highly unfamiliar environments.

The same cannot be said for autonomous driving algorithms trained using reinforcement learning (RL). In the standard RL formulation, the resulting policy π is suitable for a particular set of underlying dynamics: they are fundamentally brittle towards perturbations. This becomes especially problematic when deploying policies in the real world, where performance is significantly lower than in training due to inaccuracies in training simulation dynamics (known as the Sim-to-Real gap).

While this can be address to an extent using domain randomization (DR), random sampling of environment parameters between episodes, such policies demonstrate high variance WRT said parameters, indicating they are not truly robust to unseen conditions [1]. We therefore pursue a meta-learning approach that produces policies able to adapt online to new tracks, vehicles and opponents. To stabilize meta-updates across such heterogeneous tasks we introduce **Adaptive Beta Clipping (ABC)**—a variance-aware propensity-score clipping rule that replaces the hand-tuned constant in Meta-Q-Learning by a statistic that scales with the batch mean and variance of the importance weights. ABC reduces critic-loss variance from 2.23×10^{-6} to 8.40×10^{-7} (Sec. IV-E), yielding faster convergence without additional hyper-parameter tuning.

A. Contributions

1) We adapt the Meta-Q-Learning framework to the F1TENTH autonomous-racing environment, extending it to the multi-agent, opponent-aware setting.
2) We propose **Adaptive Beta Clipping (ABC)**, a self-tuning variance-reduction mechanism that tightens critic-loss variance by $2.7\times$, facilitating robust and efficient task adaptation.

II. Related Work

A. DDPG + TD3 Learning

Deep Deterministic Policy Gradient (DDPG) is an off-policy Q function estimation for continuous action spaces which focuses on minimizing the following mean squared Bellman error (MSBE) function:

$$\underset{\tau\sim\mathcal{D}}{E}\left[\left(Q_\theta(s_t,a_t)-(r+\gamma(1-d)\max_{a'}Q_\theta(s_{t+1},a_{t+1}))\right)^2\right] \tag{1}$$

where d denotes whether the next state s_{t+1} is terminal.

Twin-Delayed Deep Deterministic Policy Gradient (TD3) is an adaption of DDPG: a deterministic off-policy actor-critic algorithm designed to address function approximation errors in actor-critic methods. TD3 incorporates three key improvements from standard DDPG methods: clipped double Q-learning to reduce overestimation bias, delayed policy updates to stabilize learning dynamics, and target policy smoothing to regularize critics [2].

B. Meta-Q-Learning

Meta-learning is a class of algorithms which learns a network initialization θ capable of adapting to tasks sampled from a distribution $p(\mathcal{T})$ with few gradient steps. A popular example is Model-Agnostic Meta-Learning (MAML), which

finds θ maximizing performance on sampled tasks $\mathcal{T}_i \sim p(\mathcal{T})$ after a single gradient step on each task:

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i} \left(\theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(\theta)\right) \tag{2}$$

This is accomplished by first using the current θ as an initialization for task-specific θ_i, then updating θ using the aggregate objective over each $\mathcal{T}_i$:

Algorithm 1 MAML for Reinforcement Learning [3]

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Sample K trajectories $\mathcal{D} = \{(s_1, a_1, ..., s_H)\}$ using f_θ in $\mathcal{T}_i$
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    Sample trajectories $\mathcal{D}_i' = \{(s_1, a_1, ..., s_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while=0

MAML is known to suffer from computationally expensive, unstable training due to requiring second-order derivatives and using few samples for policy gradient estimation [4]. Both these issues are addressed by Meta-Q-Learning (MQL), an off-policy algorithm solving for:

$$\hat{\theta} = \operatorname*{argmax}_{\theta} \frac{1}{n} \sum_{k=1}^{n} \mathop{E}_{\tau \sim D^k} \left[\ell^k(\theta)\right] \tag{3}$$

where D^k is a set of transitions sampled from task $\mathcal{T}_k$ and ℓ^k the associated objective function [5]. A replay buffer D^m is populated using each D^k. The adaptation phase then obtains parameters θ for a new task with replay buffer D^n, objective ℓ^n by maximizing:

$$\mathop{E}_{\tau \sim D^n} [\ell^n(\theta)] + \mathop{E}_{\tau \sim D^m} [\beta(\tau; D^n, D^m) \ell^n(\theta)] - (1 - \text{ESS}) \|\theta - \hat{\theta}\|^2 \tag{4}$$

score

Here β is the propensity score, a real number which grows when training example τ has a high probability of appearing in the test distribution. Likewise, ESS uses sums of propensity scores over transitions to determine how much regularization on $\theta - \hat{\theta}$ is appropriate.[1] Details of these terms, taken from the original MQL paper, are omitted for brevity.

C. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an online reinforcement learning (RL) algorithm from the actor-critic (AC) family, known to perform well on a wide variety of environments. Actor-critic methods seek to improve the current policy π_θ by estimating the gradient with respect to the model parameters θ of the expected finite-horizon undiscounted return:

$$\nabla_\theta J(\pi_\theta) = \mathop{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) g_\theta(s_t, a_t) \right] \tag{5}$$

Here $g_\theta : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is some function describing the expected return of taking action a_t from state s_t under the current policy-our implementation utilizes the advantage function defined as:

$$A_\theta(s, a) = Q_\theta(s, a) - V_\theta(s) \tag{6}$$

Actor-critic learning consists of two jointly trained neural networks: a critic ω which estimates g_θ and an actor θ from which trajectories are sampled. The former is trained using standard MSE loss with sampled rewards as ground truth, while the latter is updated by using ω to estimate $\nabla_\theta J(\pi_\theta)$ for gradient ascent [6].

Training AC models is often quite challenging due to interactions in updates between the two networks and the possibility of large updates undoing past progress. [7] [8]. PPO addresses these issues by limiting the extent to which the policy can change between gradient steps, mitigating forgetting and ensuring the critic remains relevant for the new actor. This is accomplished by first replacing the log-probabilities of actions given states with the probability ratio $\frac{\pi_\theta(a|s)}{\pi_\phi(a|s)}$, where θ, ϕ are the new and old parameters respectively. Furthermore, a small hyperparameter ε is used to define the clipped objective:

$$L(s, a, \phi, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_\phi(a|s)} A^\phi(s, a),\ g(\varepsilon, A^\phi(s, a)) \right)$$
$$g(\varepsilon, A) = \begin{cases} (1 + \varepsilon) A & A \geq 0 \\ (1 - \varepsilon) A & A < 0 \end{cases} \tag{7}$$

$$\theta \leftarrow \operatorname*{argmax}_{\theta} \mathop{E}_{s, a \sim \pi_\phi} [L(s, a, \phi, \theta)] \tag{8}$$

Intuitively, we constrain the probability ratios to the range $(1 - \varepsilon, 1 + \varepsilon)$, ensuring π_θ has similar behavior to π_ϕ [9].

III. Methodology

A. Environment

Our experiments are conducted in a fork of the `f1tenth_gym` environment [10] modified to follow the OpenAI Gymnasium standard [11]. We use the following observation space:

```
{
    'scan': [s0...sB],
    'pos': [x,y,θ],
    'vel': [vx,vy,vz],
    'heading': [δ,β]
}
```

`'scan'` provides $B \in [0, 1080]$ uniformly spaced LiDAR beams. `'pos'` and `'vel'` contain estimated position and simulated IMU data respectively, and lastly `'heading'`

[1] In practice, Propensity Score is estimated using logistic regression, with β in turn used to estimate ESS.

measures stability via current steering and slip angles. Note that while the `env.observation_space` contains this information for all agents, we use indexing so that ego and opponent policies only access data from their own car. The action space contains $\mathbf{a} \in \mathbb{R}^{N\times 2}$ arrays where $\mathbf{a}[i]$ holds normalized (steer, speed) commands for agent $i \in [0, N]$.

Supporting opponent drivers is our primary contribution to the base environment. While `f1tenth_gym` can simulate 2 cars on track, it is configured so that a single policy control bothh vehicles. Instead, we add an `OpponentDriver` class attribute which passes the current opponent-indexed observation to a policy described in III-B. At each `step()` call, the ego and opponent actions are concatenated to exploit the original multi-agent action space: their progress along the track is then used to detect changes in track position, with overtakes yielding a large reward.

We also enable domain randomization (DR) of environment parameters such as track layout and surface friction. Numerical parameters with `min`, `max` ranges specified in the configuration YML are randomly sampled and applied at each `reset()` call. As for track layout, the `track` attribute is selected from one of the tracks in the input directory if a particular track is not specified.

Training takes advantage of Amazon EC2 `c6a.8xlarge` instances, with the environment run in headless mode. Evaluation, however, is conducted locally with GUI enabled to visually inspect agent behaviors.

B. Opponent Generation

We create opponent racers using the `stable-baselines3` [12] PPO implementation, which improves upon the basic setup described in II-C with features like gradient clipping. Using the observation and action spaces from III-A, we train single-agent policies with maximum speeds in the range $[2.0, 2.5 \dots 6.0]$ for 3 million timesteps. Each policy was manually verified as capable of completing several consecutive laps.

The reward function contains several components each encouraging fast or safe driving. Major terms include:

- *Milestones:* We partition the track into M evenly spaced checkpoints denoting $100(1/M)\%$ completion of a full lap: a large fixed reward is granted for each successive checkpoint.
- *Progress:* To mitigate sparsity issues in the milestone rewards, at each timestep we provide a reward for progress made along the centerline in the Frenet Frame.
- *Collisions:* For obstacle avoidance and general safe driving, we provide a negative reward for crashes that grows over time to a maximum penalty. We employ this curriculum approach so that early epochs are more focused on making progress along the track, preventing degenerate behaviors like standing still to avoid crashes.
- *Velocity Matching:* Another way we avoid reward hacking is by penalizing deviations from reference velocities in a pre-computed optimal trajectory.[2] We also use curriculum learning such that this penalty decreases over time, as it is only necessary to ensure that the policy learns to make progress in early episodes.
- *Action Regularization:* We include small penalties for the delta between the current and previous actions in order to encourage smooth driving.

Reward plots are included in VI-A.

[2]Optimal trajectories are calculated using [13]

C. Adaptive Beta Clipping

The original Meta-Q-Learning formulation [5] stabilizes stochastic-gradient updates by *hard-clipping* the task–propensity weights

$$\beta(\tau) = \exp\big(-f_\psi(\tau)\big) \in (0, 1], \qquad \beta \leftarrow \min\{\beta,\ \beta_{\max}\},$$

where $\beta_{\max}$ is a hand-tuned constant (typically $\beta_{\max} = 0.1$). Although effective, a global threshold suffers from two drawbacks:

- **Task heterogeneity**. The scale of β varies with the similarity between the current task D^n and the meta-replay D^m. A single $\beta_{\max}$ either clips too aggressively on "easy" tasks or too weakly on "hard" ones.
- **Hyper-parameter burden**. Choosing $\beta_{\max}$ adds an extra dimension to the tuning grid.

a) Variance-aware clipping.: We replace the fixed ceiling with a *self-tuning* threshold based on the running first- and second-order moments of β inside each mini-batch $\mathcal{B}$. Let

$$\mu_\beta^{\text{batch}} = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \beta_i^{\text{raw}}, \qquad \sigma_\beta^{2\ \text{batch}} = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} (\beta_i^{\text{raw}} - \mu_\beta^{\text{batch}})^2,$$

where $\beta_i^{\text{raw}} = \exp(-f_\psi(\tau_i))$. We maintain exponential moving averages

$$\mu_\beta^t = m\,\mu_\beta^{t-1} + (1-m)\,\mu_\beta^{\text{batch}}, \qquad \sigma_\beta^t = m\,\sigma_\beta^{t-1} + (1-m)\,\sigma_\beta^{\text{batch}},$$

with momentum $m \in [0, 1)$ (we use $m = 0.9$). Each raw weight is then clipped by

$$\beta_i \ = \ \min\big\{\ \beta_i^{\text{raw}},\ \mu_\beta^t + k\,\sigma_\beta^t\big\} \tag{9}$$

where k is a small constant; $k \in [2, 3]$ keeps roughly the upper 2–5% of a Gaussian-like tail from exploding.

b) Effect on gradient variance.: For any loss term $L = \beta\,\ell$, the update variance satisfies

$$\mathrm{Var}[L] \le \big(\mu_\beta^t + k\sigma_\beta^t\big)^2 \mathrm{Var}[\ell],$$

so the bound adapts to the current scale of β instead of relying on a pessimistic global ceiling, which should yield lower critic-loss variance without additional tuning. A proof of this property is as follows:
Let $L = \beta\,\ell$ with $\ell \in \mathbb{R}$ and denote by

$$C_t \ := \ \mu_\beta^t + k\sigma_\beta^t$$

the adaptive ceiling from Eq. (9). Clipping guarantees $\beta \le C_t$ almost surely, hence

$$|L| \ = \ |\beta\,\ell| \ \le \ C_t\,|\ell|, \qquad \text{and} \qquad L^2 \ \le \ C_t^2\,\ell^2.$$

$$\begin{aligned}\mathrm{Var}[L] &= \mathbb{E}[L^2] - \big(\mathbb{E}[L]\big)^2 \;\le\; \mathbb{E}[L^2] && \text{(dropping non-negative term)}\\ &\le C_t^2\,\mathbb{E}[\ell^2] && \text{(by bound above)}\\ &= C_t^2\,\big(\mathrm{Var}[\ell] + (\mathbb{E}[\ell])^2\big).\end{aligned}$$

In most RL objectives the per-sample contribution ℓ is mean-centered (e.g. zero-mean advantage in actor-critic, zero-mean TD error at convergence), so $\mathbb{E}[\ell] = 0$ and $\mathbb{E}[\ell^2] = \mathrm{Var}[\ell]$:

$$\mathrm{Var}[L] \;\le\; C_t^2\,\mathrm{Var}[\ell] \;=\; \big(\mu_\beta^t + k\sigma_\beta^t\big)^2\,\mathrm{Var}[\ell].$$

When $\mathbb{E}[\ell] \neq 0$ the same derivation yields $\mathrm{Var}[L] \le C_t^2\mathbb{E}[\ell^2]$, which is an even *tighter* upper bound because $\mathbb{E}[\ell^2] \ge \mathrm{Var}[\ell]$. Either way, the variance of the gradient update is controlled by the self-tuning clip level C_t, avoiding the pessimistic global constant used in the original MQL.

c) Implementation.: Algorithmically, Eq. (9) requires $\sim$20 new lines in the propensity calculation routine: compute per-batch mean/variance, update the EMA, and apply the clamp. The procedure is $O(1)$ in memory and leaves the rest of the MQL pipeline unchanged.

d) Connection to robust statistics: Adaptive clipping echoes the classical rule of thumb that observations lying more than k standard deviations from the mean are treated as outliers. Here, transitions with exceptionally low f_ψ—and correspondingly high β—are down-weighted, yielding a form of robust importance sampling that is task-agnostic and parameter-free.

D. MQL Setup

The MQL setup closely follows the implementation from [5], with much of the core functionality directly adapted from their codebase, namely the `mql.py` file, which has been instrumental in our work. All code for this project is located in the github repository `RDLigeralde/metacars` under branch `aws_deployment`. We use TD3 to train the meta-policy, and structure our actor critic networks in a similar way. We implement the actor and critic as MLPs while the context encoder is a GRU.

The context encoder is defined as a two-layer GRU with 2 hidden layers and hidden dimension 128. It processes a rolling window of 10 timesteps of historical observations, allowing the policy to condition on recent trajectories to better infer task-specific dynamics.

For each of the four components in our observation space, we define a dedicated MLP encoder. `heading_mlp`, `pose_mlp`, and `vel_mlp` consist of two 64-dimensional FC layers with ReLU activations. While `scan_mlp` uses two 128-dimensional FC layers to accommodate the higher-dimensional scan input.

In the actor network, all embeddings derived from the observation features are concatenated with GRU context embedding. This combined representation is passed through a final MLP with an output dimension of 2 and a Tanh activation to ensure the output actions fit within the target action space.

For the critic network, the architecture follows a similar pattern. The observation embeddings and context embedding are concatenated, this time along with the action input. This composite input is fed into an MLP that outputs a single scalar, the Q-value.

Following standard TD3 architecture, the critic consists of two independent Q-networks, `Q1` and `Q2`. The forward pass returns the outputs of both Q-functions, which helps mitigate overestimation bias during training by using the minimum of the two values in the TD target computation.

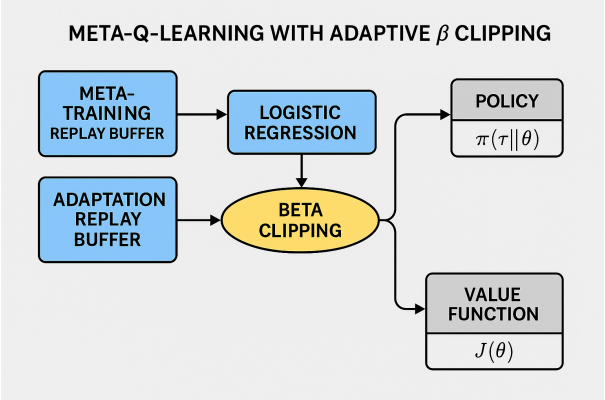


Fig. 1. Architecture, with (Adaptive) Beta Clipping

We evaluate MQL on two generalization tasks:

1) **Speed Generalization:** After training on a distribution of opponents, we assess how well the model adapts to faster opponent speeds. Specifically, the reference velocity during training is set to $8.0\,v_{\max}$. For testing, we evaluate against two held-out opponents with speeds of 6.5 and $7.0\,v_{\max}$.
2) **Track Generalization:** After training on a set of tracks: *Brands Hatch*, *IMS*, *Levine*, *Monza*, *Nürburgring*, *Yas Marina*, and *Zandvoort*, we test the model's ability to generalize to unseen tracks: *Austin*, *Budapest*, and *Catalunya*.
   We selected these hold-out tracks for two reasons:
   a) They feature a diverse set of curve geometries, which challenges the model's adaptability.
   b) Their initials form "ABC", which coincides with the acronym for the adaptive beta clipping technique that we present.

Due to limitations in parallelization and computational resources, we were only able to pre-train TD3 for 2,500 meta-iterations on an Amazon `g6.2xlarge` instance. This is grossly insufficient to produce high-quality trajectories, particularly given the off-policy nature of the algorithm. During each meta-iteration, we sample a task (either an opponent policy or a track) and perform a meta-update, adding the rollouts to both the meta buffer and the task specific buffer (both with size 512) later for adaptation. We then perform 10 adaptation iterations for each evaluation task.

All evaluation is done over 500 episodes with maximum timesteps set at 5 real time seconds since the policy has a tendency to stagnate due to low training iterations.

IV. Results

For brevity, all non-essential training metric plots can be found in the appendix.

A. Preface

We preface our results section with a reminder that our TD3 algorithm was only trained for 2,500 meta-iterations and acknowledge that any claims made may be the result of randomness and hold very little (if any) statistical significance.

B. Opponent Adaptation

We focus our discussion on how well policies do vs out-of-sample opponents.

Metric	Vanilla DR	MQL	MQL-ABC
Overtakes	135	0	0
Crashes	395	500	500
Mean Ep. Len	160.83 ± 195.87	27.85 ± 29.49	25.39 ± 28.12
Mean Reward	-0.621 ± 0.487	-1.018 ± 0.023	-1.212 ± 0.012

TABLE I
OUT OF SAMPLE OPPONENTS PERFORMANCE

We generally see that vanilla DR is able to successfully overtake whereas both MQL variant struggle a bit with that; furthermore, since ABC tends to further reduce episode length and reward, this suggests potential signs of overfitting.

C. Map Adaptation In Distribution

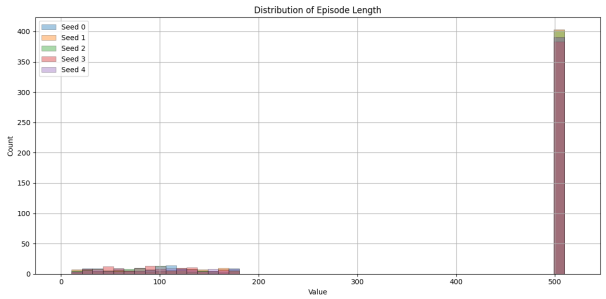
For in distribution tasks, there

Metric	Vanilla DR	MQL
Crashes	211	93
Episode Length	393.22 ± 167.19	423.80 ± 183.25
Reward	-0.308 ± 0.4641	-0.1934 ± 0.3821

TABLE II
IN DISTRIBUTION MAP ADAPTATION

The table above is for in-sample results. On tracks seen during meta-training, both Vanilla DR and MQL maintain relatively longer episode lengths, indicating successful navigation without crashing. MQL also attains a higher average reward and fewer crashes, indicating a learned initialization that is able to navigate familiar layouts well. Since MQL beta clipping does not really come in until the adaptation step, MQL and MQL-ABC are one and the same here.

As reflected by the persistently negative mean reward, the PPO policy is suboptimal on more complex track geometries. Under harder tracks, non-recurrent linear layers don't capture the different curves as well and often lead to longer episodes with slow progress. This is also characterized by the figure below: we show that the distribution of episode length is effectively bimodal: the car either crashes or times out. These difficulties could also be due to the fact that training was halted after a few meta-iterations, so the stagnation-avoidance curriculum had not yet warmed up.



D. Map Adaptation Out of Distribution

Metric	Vanilla DR	MQL	MQL-ABC
Crashes	157	183	203
Ep. Len	423.84 ± 125.38	326.2 ± 213.26	307.22 ± 134.12
Reward	-1.12 ± 0.6754	-0.403 ± 0.4920	-0.4516 ± 0.234

TABLE III
MAP ADAPTATION OUT OF DISTRIBUTION

When adapting to unseen tracks, MQL appears to outperform vanilla DR in terms of average rewards and smaller episode lengths. Under DR, a majority of the episodes end in the car standing still or running into a wall, explaining the high mean episode length. MQL-ABC has slightly lower rewards than the standard MQL but reduces episode length and thus avoids many useless episodes.

E. Adaptive Beta Clipping

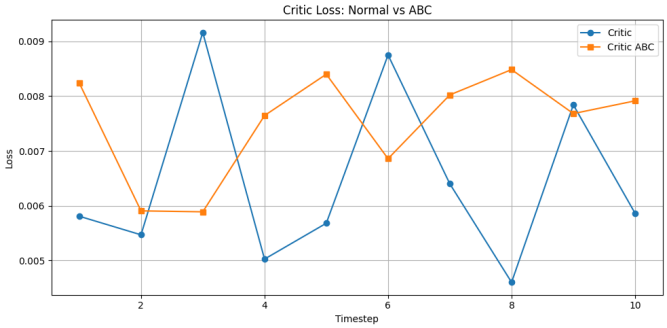


Fig. 2. Adaptive Beta Clipping (ABC) Ablation

While it is difficult to meaningfully compare task adaption with the current setup, we do see that critic loss does become tighter with the implementation of ABC. More specifically, the variance drops from 2.23×10^{-6} to 8.4×10^{-7}, a $2.7\times$ reduction.

V. Conclusion + Future Work

Despite unremarkable outcomes we still believe that this method has strong potential. Below we describe some ways in which stronger results could be achieved.

A. More Efficient MQL Implementation

A major bottleneck in our training process was that the provided implementation of [5] was incompatible with vectorized environments from `stable-baselines3`. As a result, we

were severely limited in the size of our replay buffer, which contributed to the sub-par TD3 results described in Section IV. We had originally considered evaluating how well a policy adapted for a single task would perform compared to the meta-policy; however, given the poor performance of TD3 under these constraints, this comparison proved uninteresting. Given more time, we would have reimplemented the MQL framework with full compatibility for `stable-baselines3`, enabling significantly more trajectories to be collected and improving overall learning stability.

B. Network Architecture Improvements

We also believe significant improvements can be made in the network architecture. The default SB3 feature extractor uses simple MLPs for each observation dictionary key, but since LiDAR data contains strong local dependencies similar to images, applying convolutional layers to `'scan'` should be beneficial. While our own experiments here were unsuccessful, we believe this is still a promising direction.

REFERENCES

[1] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," *CoRR*, vol. abs/1904.04762, 2019. [Online]. Available: http://arxiv.org/abs/1904.04762

[2] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[3] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *CoRR*, vol. abs/1703.03400, 2017. [Online]. Available: http://arxiv.org/abs/1703.03400

[4] H. Liu, R. Socher, and C. Xiong, "Taming MAML: Efficient unbiased meta-reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 4061–4071. [Online]. Available: https://proceedings.mlr.press/v97/liu19g.html

[5] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola, "Meta-q-learning," *arXiv preprint arXiv:1910.00125*, 2019.

[6] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[7] S. Parisi, V. Tangkaratt, J. Peters, and M. E. Khan, "Td-regularized actor-critic methods," *CoRR*, vol. abs/1812.08288, 2018. [Online]. Available: http://arxiv.org/abs/1812.08288

[8] T. Zhang, X. Wang, B. Liang, and B. Yuan, "Catastrophic interference in reinforcement learning: A solution based on context division and knowledge distillation," *CoRR*, vol. abs/2109.00525, 2021. [Online]. Available: https://arxiv.org/abs/2109.00525

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[10] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, ser. Proceedings of Machine Learning Research, H. J. Escalante and R. Hadsell, Eds., vol. 123. PMLR, 08–14 Dec 2020, pp. 77–89. [Online]. Available: https://proceedings.mlr.press/v123/o-kelly20a.html

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

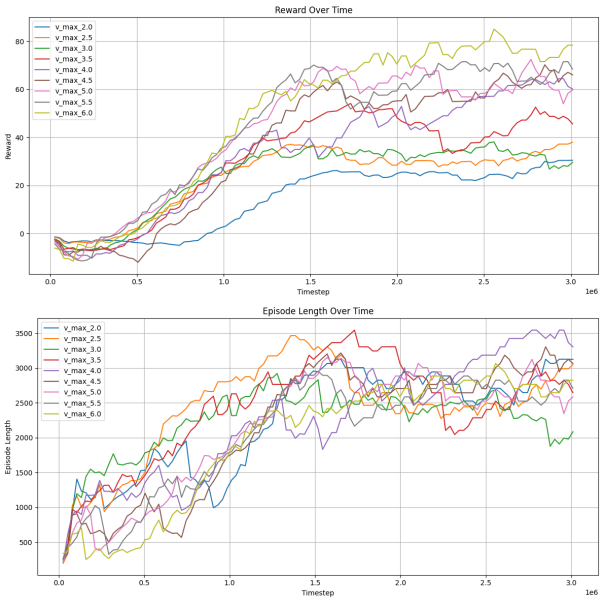
[12] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[13] F. Christ, A. Wischnewski, A. Heilmeier, and B. Lohmann, "Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients," *Vehicle System Dynamics*, vol. 59, no. 4, pp. 588–612, 2021, publisher Copyright: © 2019 Informa UK Limited, trading as Taylor Francis Group.

VI. Appendix

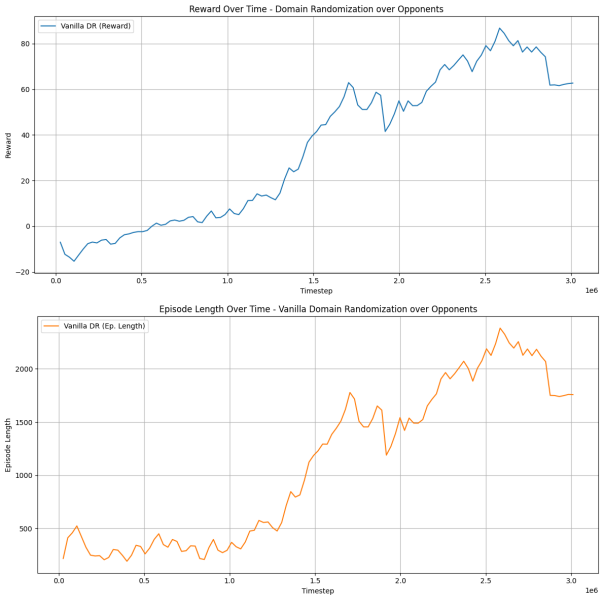
A. PPO training for Opponent Generation

Below are plots showing the training performance of agents using PPO across different `v_max` settings. Around 1.5 million timesteps, we observe that the reward plateaus and no longer shows significant improvement. We see that agents with higher maximum speed achieve higher reward.



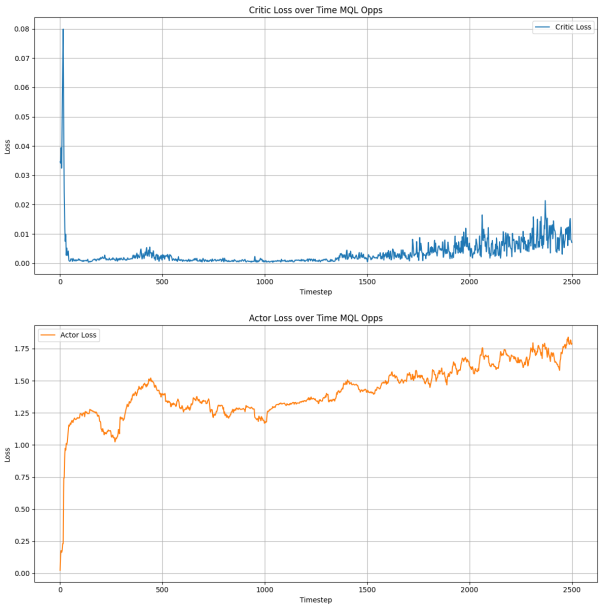
B. PPO training for Opponents

Below are plots showing the training performance vanilla domain randomization against different opponent cars.

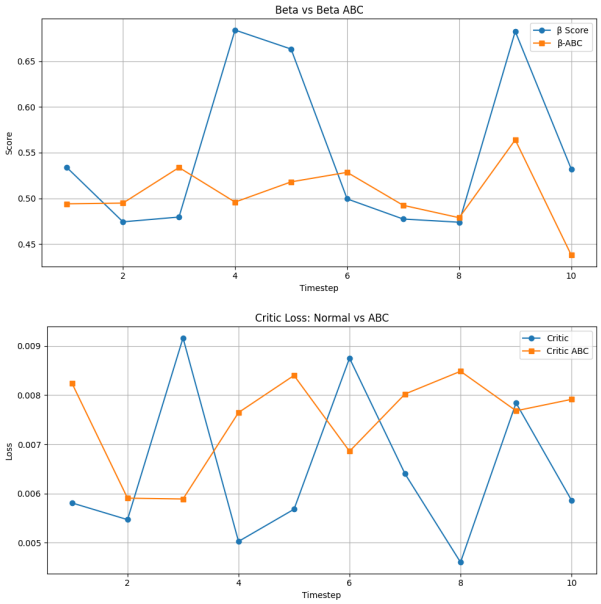


C. MQL Training over Opponents

Plots below show that critic loss converges whereas actor loss remains non-zero/persistent.

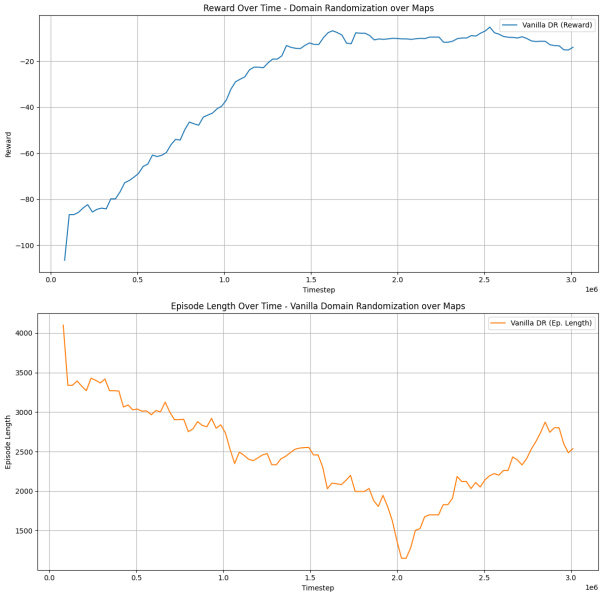


D. β vs β_{ABC} for Opponents During Adaptation



E. Vanilla PPO Domain Randomization over Tracks

The reward here remains negative throughout; this would likely be because the car is unable to localize accurately/maintain sufficient progress before incurring the stagnation penalty (i.e. the agent dies out due to minimal forward movement). A potential solution would be increasing the replay buffer size and extending the observation history window for observation, state, etc.



F. MQL over Tracks

This is a lot more erratic than the opponent loss. This is probably because there is a lot more variation with different tracks compared to same track different speed opponents.