
DEEP REINFORCEMENT LEARNING

HOMEWORK 8 | MULTI-AGENT RL, META-RL

Professor Mohammad Hossein Rohban

Sharif University of Technology
Spring 2026



Contents

1	Meta-RL through Gradient-Based Adaptation	1
2	Learning Dynamics in Two-Player Matrix Games	4
3	MADDPG and MAPPO under Centralized Training	7
4	Meta-Reinforcement Learning on HalfCheetahVel	10

Important Notes

- **Grading Policy:** This homework is considered complete once you reach a total of **100 points**. Up to *5 additional bonus points* may be earned; however, any score above 100 will still be recorded as 100. Bonus points do *not* transfer or overflow to other assignments.
- **Homework Deadline:** The assignment is due on the posted deadline. Please make sure to submit your work on time.
- **Delay Usage Policy:** You may use *at most 5 days* of your allowed late-day budget for this homework. Late days will be automatically applied if you submit after the deadline.
- **Cheating and Collaboration Policy:** You must write all solutions in your own words and produce all code independently. Discussing general ideas is allowed, but sharing written solutions, code, or using someone else's work (including from online sources, previous semesters, or AI tools without proper authorization) is strictly prohibited. Violations will result in academic misconduct procedures.

Advice

Please avoid asking LLMs for the answer. Sure, doing so might give you an immediate reward of 5 points, but in the long run, it won't help you become the next Rockstar of DRL. Remember the key lesson of this course: chasing immediate rewards is rarely the best strategy. Instead, struggle with the problems, read the papers, and truly understand the algorithms. The satisfaction of solving a challenging problem yourself is far greater than any grade. Good luck! [?]

Grading Breakdown

Task / Criteria	Points
Meta-RL through Gradient-Based Adaptation	21 pts
Exact Bilevel Differentiation and Inner-Loop Unrolling	7 pts
Stochastic Policy-Gradient Curvature	7 pts
First-Order Approximation and Trajectory Distribution Shift	7 pts
Learning Dynamics in Two-Player Matrix Games	32 pts
Empirical-Belief Fictitious Play	7 pts
Exploratory Fictitious Play	7 pts
Regret Matching and No-Regret Diagnostics	7 pts
Mixed-Strategy Nash Equilibria	6 pts
Convergence, Exploration, and Regret	5 pts
MADDPG and MAPPO under Centralized Training	39 pts
MADDPG Actor, Centralized Critic, and Off-Policy Update	9 pts
MAPPO Policy, GAE, and Clipped Optimization	9 pts
Controlled CTDE Comparison	9 pts
Stability versus Sample Efficiency	6 pts
Centralization, Parameter Sharing, and Credit Assignment	6 pts
Meta-Reinforcement Learning on HalfCheetahVel	33 pts
Shared Infrastructure and Differentiable MAML	9 pts
Recurrent Fast Adaptation with RL ²	9 pts
Latent Task Inference with VariBAD	9 pts
Three Mechanisms for Fast Adaptation	6 pts
Bonus	+5 pts
Clarity and Quality of Report and Codes	+5 pts
Total	125+5 pts

1 Meta-RL through Gradient-Based Adaptation

Meta-reinforcement learning seeks an initialization from which a policy can adapt to a previously unseen task using only a small amount of task-specific experience. Let $\mathcal{M} \sim p(\mathcal{M})$ be a finite-horizon MDP drawn from a task distribution. Tasks share state and action spaces but may differ in their reward functions or transition dynamics.

Scope. This problem is restricted to exact and first-order differentiation through a fixed number of policy-gradient adaptation steps. The inner step sizes are fixed, and only the policy parameters are adapted. Implicit differentiation, learned step sizes, shared representation parameters, recurrent task inference, off-policy trajectory reuse, learned baselines, and finite-sample bias in the inner update are outside the scope of the problem.

Trajectory model. For policy parameters ϑ , let a trajectory have density

$$p_{\mathcal{M},\vartheta}(\tau) = \rho_{\mathcal{M}}(s_0) \prod_{t=0}^{T-1} \pi_{\vartheta}(a_t | s_t) P_{\mathcal{M}}(s_{t+1} | s_t, a_t).$$

The initial-state distribution, transition kernel, and trajectory return have no direct dependence on ϑ . Thus, all parameter dependence of the trajectory distribution enters through the policy.

For inner step k , define the population support loss

$$L_{\mathcal{M},k}^{\text{tr}}(\vartheta) = -\mathbb{E}_{\tau \sim p_{\mathcal{M},\vartheta}} [R_{\mathcal{M},k}^{\text{tr}}(\tau)],$$

and define the population query loss

$$L_{\mathcal{M}}^{\text{val}}(\vartheta) = -\mathbb{E}_{\tau \sim p_{\mathcal{M},\vartheta}} [R_{\mathcal{M}}^{\text{val}}(\tau)].$$

The support loss may differ across inner steps.

Adaptation and outer objective. Starting from the shared initialization θ , define

$$\theta_{\mathcal{M}}^{(0)} = \theta, \quad \theta_{\mathcal{M}}^{(k+1)} = \theta_{\mathcal{M}}^{(k)} - \alpha_k \nabla L_{\mathcal{M},k}^{\text{tr}}(\theta_{\mathcal{M}}^{(k)}), \quad k = 0, \dots, K-1.$$

The task-level and expected outer objectives are

$$f_{\mathcal{M}}(\theta) = L_{\mathcal{M}}^{\text{val}}(\theta_{\mathcal{M}}^{(K)}), \quad F(\theta) = \mathbb{E}_{\mathcal{M} \sim p(\mathcal{M})} [f_{\mathcal{M}}(\theta)].$$

The inner updates use exact population gradients. Consequently, conditional on $\mathcal{M}$ and θ , the adaptation path $\theta_{\mathcal{M}}^{(0)}, \dots, \theta_{\mathcal{M}}^{(K)}$ is deterministic. Trajectory samples are introduced only to construct Monte Carlo estimators of the derivatives of this population objective.

Derivative conventions and regularity. All gradients are column vectors. For a vector-valued function $y(x)$, the Jacobian $\partial y / \partial x$ has dimensions $\dim(y) \times \dim(x)$. Assume a common dominating measure for the trajectory distributions, parameter-independent trajectory support, twice continuously differentiable log-policies, and integrable dominating envelopes for

$$|R(\tau)| \|s_{\vartheta}(\tau)\|_2, \quad |R(\tau)| \|s_{\vartheta}(\tau) s_{\vartheta}(\tau)^{\top} + \nabla_{\vartheta}^2 \log p_{\mathcal{M},\vartheta}(\tau)\|_2,$$

together with the corresponding task-level meta-gradient products. These assumptions permit differentiation through both trajectory and task expectations.

Algorithm 1: Reverse-Mode Estimation of the Exact Meta-Gradient

- 1: Sample a task $\mathcal{M}$ and form the population adaptation path $\theta_{\mathcal{M}}^{(0)}, \dots, \theta_{\mathcal{M}}^{(K)}$.
- 2: Collect an independent query batch using $\pi_{\theta_{\mathcal{M}}^{(K)}}$ and construct an unbiased estimator $\hat{v}_K$ of $\nabla L_{\mathcal{M}}^{\text{val}}(\theta_{\mathcal{M}}^{(K)})$.
- 3: **for** $k = K - 1, \dots, 0$ **do**
- 4: Collect a fresh support batch using $\pi_{\theta_{\mathcal{M}}^{(k)}}$, independent of $\hat{v}_{k+1}$ and of all other support and query batches.
- 5: Construct a higher-order-correct unbiased estimator $\hat{\mathcal{H}}_{\mathcal{M},k}(\hat{v}_{k+1})$ of

$$\nabla^2 L_{\mathcal{M},k}^{\text{tr}}(\theta_{\mathcal{M}}^{(k)})^\top \hat{v}_{k+1},$$
 treating $\hat{v}_{k+1}$ as constant during differentiation.
- 6: Set

$$\hat{v}_k \leftarrow \hat{v}_{k+1} - \alpha_k \hat{\mathcal{H}}_{\mathcal{M},k}(\hat{v}_{k+1}).$$
- 7: **end for**
- 8: Use $\hat{v}_0$ as the task-level exact meta-gradient estimator.

Question 1: Exact Bilevel Differentiation and Inner-Loop Unrolling**[7 Points]**

Assume all conditional moments appearing below are finite. For a sampled task $\mathcal{M}$, define

$$\theta'_{\mathcal{M}} = \theta - \alpha \nabla_{\theta} L_{\mathcal{M}}^{\text{tr}}(\theta), \quad F(\theta) = \mathbb{E}_{\mathcal{M}} [L_{\mathcal{M}}^{\text{val}}(\theta'_{\mathcal{M}})].$$

- (a) Derive the exact one-step meta-gradient $\nabla_{\theta} F(\theta)$. Write the result using the train-loss Hessian and the validation-loss gradient evaluated at $\theta'_{\mathcal{M}}$, with all evaluation points and matrix transposes explicit.
- (b) Now let

$$\theta^{(0)} = \theta, \quad \theta^{(k+1)} = \theta^{(k)} - \alpha_k \nabla L_{\mathcal{M},k}^{\text{tr}}(\theta^{(k)}).$$

Derive the ordered-product expression for

$$J_K = \frac{\partial \theta^{(K)}}{\partial \theta},$$

and use it to express the exact K -step meta-gradient.

Question 2: Stochastic Policy-Gradient Curvature**[7 Points]**

For a fixed task, let

$$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}} [R(\tau)], \quad g(\tau) = \nabla_{\theta} \log p_{\theta}(\tau), \quad S(\tau) = \nabla_{\theta}^2 \log p_{\theta}(\tau),$$

where $R(\tau)$ has no direct dependence on θ .

- (a) Starting from the likelihood-ratio identity

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}} [R(\tau) g(\tau)],$$

derive $\nabla_{\theta}^2 J(\theta)$. Your expression must display separately the score outer-product term and the log-density Hessian term.

- (b) For a fixed vector v , derive an unbiased single-trajectory estimator of

$$\nabla_{\theta}^2 J(\theta) v$$

that does not explicitly construct the Hessian. Identify the term that would be omitted by differentiating the sampled vector $R(\tau)g(\tau)$ while treating the sampled trajectory as fixed.

Question 3: First-Order Approximation and Trajectory Distribution Shift

[7 Points]

For K inner steps, define

$$A_k = I - \alpha_k H_k, \quad J_K = A_{K-1} \cdots A_0, \quad q = \nabla_{\theta^{(K)}} L_{\mathcal{M}}^{\text{val}}(\theta^{(K)}).$$

The exact and first-order task gradients are

$$g_{\text{exact}} = J_K^{\top} q, \quad g_{\text{FO}} = q.$$

- (a) Assume

$$\|H_k\|_2 \leq H, \quad 0 \leq \alpha_k \leq \bar{\alpha}, \quad \|q\|_2 \leq G.$$

Derive a bound on

$$\|g_{\text{exact}} - g_{\text{FO}}\|_2$$

in terms of H , G , and the inner step sizes. Then specialize the result to the constant upper bound $\alpha_k \leq \bar{\alpha}$.

- (b) Let $p_{\theta}(\tau)$ and $p_{\theta'}(\tau)$ denote the pre-adaptation and post-adaptation trajectory distributions in the same MDP. Derive the importance-sampling identity that expresses

$$\mathbb{E}_{\tau \sim p_{\theta'}}[R(\tau)]$$

using trajectories sampled from p_{θ} . Write the weight as a product of policy-probability ratios over time, and characterize how support mismatch and trajectory horizon affect its variance.

2 Learning Dynamics in Two-Player Matrix Games

This notebook introduces multi-agent learning through two zero-sum variants of a Cavalry–Infantry–Archers game. The standard game has the cyclic dominance structure of rock–paper–scissors: Cavalry defeats Archers, Archers defeat Infantry, and Infantry defeats Cavalry. A modified payoff matrix changes the magnitudes of these advantages while preserving strategic interaction. Unlike a single-agent problem, neither player’s learning target is stationary because the opponent’s behaviour changes at the same time.

A mixed-strategy Nash equilibrium is a pair of probability distributions for which neither player can improve their expected payoff by changing strategy alone. In a fully mixed equilibrium, every pure action assigned positive probability must have the same expected payoff. These indifference conditions, together with non-negativity and normalization, provide a linear system for the equilibrium probabilities. The equilibrium is used as a reference point for the empirical learning dynamics below.

In fictitious play, each player treats the opponent’s empirical action frequencies as a stationary belief. If $C_{t-1}(a_{-i})$ is the number of times the opponent has selected action a_{-i} , and $\hat{\mu}_{-i,t-1}(a_{-i}) = C_{t-1}(a_{-i})/(t-1)$ for $t \geq 2$, player i chooses

$$a_{i,t} \in \arg \max_{a_i \in A_i} \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \hat{\mu}_{-i,t-1}(a_{-i}), \quad t \geq 2,$$

When several actions are best responses, the implementation samples uniformly from the tied set. Fictitious play is deterministic apart from initialization and tie-breaking and therefore explores only when its changing beliefs make a different action optimal.

The ϵ -greedy variant adds explicit exploration. At each round, a player selects a uniformly random action with probability ϵ and a best response to the opponent’s empirical strategy with probability $1 - \epsilon$. This prevents purely exploitative lock-in, but persistent exploration also means that the observed action distribution need not equal the unperturbed equilibrium exactly.

Regret matching replaces best responses with counterfactual regret. After player i selects a_i against a_{-i} , the cumulative regret for an alternative s is updated by

$$R_t(s) = R_{t-1}(s) + u_i(s, a_{-i}) - u_i(a_i, a_{-i}).$$

The next strategy is proportional to positive regret,

$$p_{t+1}(s) = \frac{R_t^+(s)}{\sum_{s'} R_t^+(s')}, \quad R_t^+(s) = \max\{0, R_t(s)\},$$

with a uniform strategy when all positive regrets are zero. Instantaneous strategies can oscillate even when time-averaged strategies become stable; consequently, both quantities and the average regret should be monitored.

Implementation 1: Empirical-Belief Fictitious Play

[7 Points]

Complete `simulate_fictitious_play` for arbitrary compatible payoff matrices.

1. Validate the matrix shapes, initialize each player’s action counts without creating a zero-denominator belief, and store the empirical frequency of every action at every iteration.
2. Compute all pure-action expected payoffs against the opponent’s empirical distribution. Select uniformly from all actions tied for the maximum rather than relying on the first index returned by `argmax`.
3. Run one million iterations for both games. Plot each player’s action frequencies and the distance

from the corresponding theoretical Nash strategy. Use fixed random seeds for reproducibility and ensure that the final history row contains the final empirical frequencies.

Implementation 2: Exploratory Fictitious Play

[7 Points]

Complete `simulate_epsilon_greedy_fp` so that each player independently explores with probability ϵ and otherwise selects a uniformly tie-broken best response.

1. Update action counts and empirical beliefs in a clearly documented order. Compute and record Player 1's expected payoff under the empirical joint strategy at every iteration.
2. On the modified game, run one million iterations for $\epsilon \in \{0.01, 0.1, 0.3\}$ using comparable random seeds. Plot both players' empirical frequencies, the theoretical Nash reference lines, and the payoff trace.
3. Report the final strategy, distance to equilibrium, and mean payoff over a declared tail window for each value of ϵ .

Implementation 3: Regret Matching and No-Regret Diagnostics

[7 Points]

Complete `simulate_regret_matching` for both players.

1. At each round, form a valid strategy from positive cumulative regrets, sample each player's action, and update the regret of every counterfactual action using the action actually taken by the opponent.
2. Store instantaneous strategies, running-average strategies, and Player 1's sum of positive cumulative regrets. Carefully handle the all-nonpositive-regret case with a uniform distribution.
3. Run the algorithm on the modified game and plot instantaneous and average strategies separately. In addition to cumulative positive regret, plot a per-round regret diagnostic such as $\sum_s R_t^+(s)/t$, because sublinear cumulative regret is assessed after normalization by time.

Question 1: Mixed-Strategy Nash Equilibria

[6 Points]

Derive the mixed-strategy Nash equilibrium of both the standard and modified Cavalry–Infantry–Archers games.

1. Let (q_C, q_I, q_A) denote Player 2's strategy. Write Player 1's expected payoff for each pure action and solve the indifference equations together with $q_C + q_I + q_A = 1$. Repeat symmetrically for Player 1.
2. Verify that the resulting distributions are valid and that no pure unilateral deviation gives either player a strictly larger expected payoff. State the equilibrium value of each zero-sum game.
3. Explain why changing payoff magnitudes can change equilibrium probabilities even when the cyclic win/loss ordering remains unchanged.

Question 2: Convergence, Exploration, and Regret

[5 Points]

Use your simulations to compare the three learning rules.

1. Distinguish convergence of the action selected at a particular round, convergence of the instantaneous mixed strategy, and convergence of the empirical average strategy. Which of these

notions is supported by each of your plots?

2. Explain how increasing ϵ changes transient learning, long-run action frequencies, payoff variance, and distance to the unperturbed Nash equilibrium. Derive the exploratory mixture one would expect if the exploitative component settled on a fixed distribution.
 3. Determine whether the regret-matching run provides evidence of no-regret learning. Discuss why cumulative positive regret need not decrease, why normalization by time matters, and how average strategies may approach equilibrium while instantaneous strategies continue to cycle.
- Support every claim with a quantitative result or a clearly identified feature of a plot.

3 MADDPG and MAPPO under Centralized Training

This notebook applies two multi-agent deep reinforcement-learning algorithms to the continuous-control VMAS Navigation task with three agents and vectorized parallel environments. Each agent observes local information and selects a bounded continuous action. Both algorithms use centralized training with decentralized execution (CTDE): training-time critics may use information from all agents, while each deployed actor uses only its own observation. Parameter sharing among homogeneous actors improves sample reuse but does not give an actor access to other agents' observations at execution time.

MADDPG is an off-policy deterministic actor–critic. A shared decentralized actor $\mu_\theta(o_i)$ produces agent i 's action, with external Gaussian noise added during data collection for exploration. A centralized critic estimates a joint action value from all observations and actions,

$$Q_\phi(o_1, \dots, o_n, a_1, \dots, a_n).$$

Transitions are reused from a replay buffer. The critic minimizes a temporal difference error using slowly moving target networks, while the actor follows the deterministic policy gradient through the critic. Soft target updates reduce harmful feedback between the learned target and the current networks.

MAPPO is an on-policy stochastic actor–critic. Each decentralized actor emits the location and scale of a squashed Gaussian policy. A centralized value function $V_\phi(o_1, \dots, o_n)$ supplies lower-variance baselines and GAE advantages. The actor is trained with the PPO clipped surrogate,

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

where ρ_t is the new-to-behaviour policy probability ratio. A value loss and entropy bonus are optimized with the policy objective. Because MAPPO is on-policy, each collected batch is used for only a bounded number of epochs before new experience is gathered.

The comparison exposes a central trade-off. MADDPG can be sample efficient because it reuses transitions, but deterministic policies, replay generated by changing co-agents, and bootstrapped critics can make optimization brittle. MAPPO usually produces more conservative updates and stable advantages, but it requires fresh on-policy interaction. The notebook's MADDPG/IDDPG and MAPPO/IPPO switches isolate the effect of critic centralization while leaving the actors decentralized.

Implementation 4: MADDPG Actor, Centralized Critic, and Off-Policy Update

[9 Points]

Complete the MADDPG portion of the TorchRL notebook.

1. Construct the deterministic ProbabilisticActor from the shared local-observation policy module using TanhDelta and the environment action bounds. Add Gaussian exploration only in the collector policy, not in the actor optimized by the loss.
2. Build the observation–action concatenation module and centralized critic with the required TensorDict keys. Demonstrate from tensor shapes that the critic sees joint information while a deployed actor sees only one agent's local observation.
3. Complete each replay-buffer update: sample a minibatch, compute actor and critic losses, backpropagate their sum, clip gradients, step and clear both optimizers, and soft-update target networks. Log losses, exploration scale, episodic return, and environment frames.

Implementation 5: MAPPO Policy, GAE, and Clipped Optimization**[9 Points]**

Complete the MAPPO portion of the notebook.

1. Construct the stochastic actor from the Gaussian location and scale outputs using `TanhNormal`. Store bounded actions and their log probabilities under the exact `TensorDict` keys required by PPO.
2. Create `ClipPPOLoss` with the centralized value network, configured clipping and entropy coefficient, and `normalize_advantage=False`. Compute GAE after expanding episode termination flags to the agent dimension, and verify the shapes of returns, values, advantages, and log probabilities.
3. For each on-policy batch, sample without replacement across the requested epochs, combine policy, critic, and entropy loss terms, apply gradient clipping, update the optimizer, and synchronize the collector's policy weights. Log the approximate KL divergence or probability-ratio statistics in addition to return and loss components.

Implementation 6: Controlled CTDE Comparison**[9 Points]**

Run a controlled comparison of MADDPG, IDDPG, MAPPO, and IPPO.

1. Use multiple random seeds, identical VMAS task settings, and clearly reported interaction budgets. Do not compare methods only by training iteration when their frames per iteration differ.
2. Produce learning curves against environment frames with a mean and uncertainty band across seeds. Evaluate each final policy without training exploration noise and report both team return and per-agent return.
3. Render at least one representative evaluation episode for each algorithm family and describe one observed coordination success or failure.

Question 1: Stability versus Sample Efficiency**[6 Points]**

Explain the empirical differences between MADDPG and MAPPO.

1. Why can MAPPO learn more smoothly or rapidly early in training even though MADDPG reuses experience? Discuss stochastic exploration, clipped updates, GAE, bootstrapping error, and multi-agent non-stationarity.
2. Why can MADDPG nevertheless be more sample efficient when training is stable? Distinguish environment interactions from gradient updates and explain the benefit and cost of replay.
3. Use logged quantities from your runs to diagnose at least one unstable period. Relate changes in critic loss, gradient norm, entropy or exploration scale, and return to a plausible mechanism rather than relying only on the final curve.

Question 2: Centralization, Parameter Sharing, and Credit Assignment**[6 Points]**

Analyze the role of centralized critics and decentralized actors.

1. Distinguish IDDPG from MADDPG and IPPO from MAPPO. For each pair, specify exactly which inputs change, which components remain shared or decentralized, and whether execution-time information requirements change.
2. Explain the failure modes that can arise when MAPPO's critic is made fully decentralized

while its policy remains stochastic and shared. Address partial observability, credit assignment, increased advantage variance, and symmetry between homogeneous agents.

3. Compare your centralized- and decentralized-critic ablations. Does centralization improve mean performance, variance across seeds, coordination, or all three? Identify at least one confound that prevents a learning-curve difference from proving that CTDE alone caused the result.

4 Meta-Reinforcement Learning on HalfCheetahVel

This notebook studies three approaches to meta-reinforcement learning on HALFCHEETAHVEL: MAML, RL², and VariBAD. A task is specified by a target forward velocity $v^* \in [0, 3]$, but the target is hidden from the agent. At time step t , the common reward is

$$r_t = -|v_t - v^*| - 0.05\|a_t\|_2^2,$$

where v_t is the measured forward velocity and a_t is the action. An episode lasts $H = 200$ steps. Training, validation, and test velocities must be disjoint, and neither the target velocity nor a task identifier may be given to a policy, value function, or inference network.

The algorithms differ primarily in how they perform fast adaptation. MAML learns policy parameters that can be adapted by a small number of policy-gradient steps. For task $\mathcal{T}_i$, support trajectories produce differentiable fast parameters

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}^{\text{support}}(\theta),$$

and fresh query trajectories collected with $\pi_{\theta'_i}$ define the outer objective used to update θ . Thus, MAML stores prior knowledge in an initialization and adapts through gradient descent. The method required here is vanilla MAML rather than E-MAML.

In RL², the fast learning procedure is represented by the hidden state of a recurrent actor–critic. At each step, the recurrent network receives the current observation together with the previous action, reward, and episode-boundary indicator. Two episodes from the same task form one 400-step meta-RL trial. The physical environment is reset between the two episodes, but the recurrent hidden state is not. The slowly learned network weights determine how the hidden state uses early experience to change later behaviour.

VariBAD instead treats adaptation as approximate inference in a Bayes-adaptive MDP. A recurrent encoder maintains a posterior belief $q_{\phi}(m \mid \tau_{:t})$ over a latent task variable m . A stochastic policy and value function condition on both the physical state and this belief. For HalfCheetahVel, a reward decoder trains the latent representation with a variational objective such as

$$\mathcal{L}_{\text{ELBO},t} = \mathbb{E}_{q_{\phi}(m \mid \tau_{:t})}[\log p_{\psi}(r_t \mid s_t, a_t, m)] - D_{\text{KL}}(q_{\phi}(m \mid \tau_{:t}) \parallel p(m \mid \tau_{:t-1})).$$

The posterior is initialized to the prior at a new task, updated after each transition, and retained across episode boundaries within that task.

All three methods are evaluated on the same held-out tasks. The first-rollout return measures behaviour before useful task-specific adaptation, while the adapted return is obtained from the second rollout for RL² and VariBAD and from a fresh query rollout after MAML support-set adaptation. A second evaluation follows performance across five consecutive rollouts while retaining only the method-specific adaptation state permitted by the protocol.

Implementation 7: Shared Infrastructure and Differentiable MAML

[9 Points]

Implement the common benchmark and the MAML pipeline.

1. Complete the trajectory representation, rollout collection, and GAE computation. Correctly distinguish true termination from time-limit truncation, bootstrap at a time limit, and document all tensor shapes.
2. Implement a continuous-action stochastic MAML policy, a value baseline, and a functional forward pass that can use task-specific fast parameters. The inner update must remain differentiable when exact MAML is selected.
3. Implement support collection, one or more inner policy-gradient updates, fresh query collection,

and the outer meta-update over a batch of training tasks. Report first-rollout and post-adaptation query returns in the notebook's required history format, and store the relevant hyperparameters in its metadata.

Your implementation must never expose the target velocity to the agent and must not use validation or test tasks for gradient updates.

Implementation 8: Recurrent Fast Adaptation with RL²

[9 Points]

Implement a recurrent Gaussian actor–critic whose input contains the current observation, previous action, previous reward, and previous done/reset flag. The network must return an action distribution, a value prediction, and its next recurrent state.

1. Collect ordered two-episode trials. Reset the recurrent state only when the task changes; reset the physical simulator, preserve the hidden state, and provide the boundary indicator when the first episode ends.
2. Implement PPO or TRPO updates over complete recurrent sequences (or correctly initialized contiguous subsequences). Do not randomly shuffle individual time steps or reuse stale hidden states.
3. Implement training, checkpoint evaluation, and five-rollout test evaluation. Return histories and per-rollout arrays with the exact shapes required by the provided validation and plotting functions.

Implementation 9: Latent Task Inference with VariBAD

[9 Points]

Implement the inference and control components of VariBAD.

1. Build a recurrent encoder that maps transition history to posterior mean and variance, supplies an initial prior, and produces latent samples by reparameterization. Preserve the recurrent belief across the physical reset between episodes of a task.
2. Implement the reward decoder and ELBO, including reward reconstruction and KL terms. State the sign convention and weighting used in the optimized VAE loss, and log both components separately.
3. Implement the latent-conditioned actor and value function, the on-policy control update, and the VAE update. Complete the training and evaluation loops so that first-rollout, adapted, and five-rollout results are generated from trained checkpoints rather than manually entered data.

Question 1: Three Mechanisms for Fast Adaptation

[6 Points]

Compare MAML, RL², and VariBAD as mechanisms for adapting to an unobserved target velocity.

1. For each method, identify where task-specific information is stored during meta-test time, what operation updates that information, and exactly what must be reset when moving to a new task.
2. Explain how each method can trade off information gathering in the first rollout against reward maximization in later rollouts. Which method makes uncertainty about the task most explicit, and why does this not by itself guarantee the best return?
3. Predict the qualitative failure caused in each method by accidentally resetting its adaptation state between two episodes of the same task. Relate your prediction to the first-rollout and

adapted-return curves.

Support the discussion with evidence from your trained models, such as learning curves, adaptation gains, posterior statistics, or controlled reset ablations.

References

- [1] J. Hu and M. P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 2003.
- [2] R. Lowe et al. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. NeurIPS, 2017.
- [3] C. Yu et al. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. NeurIPS Datasets and Benchmarks, 2022.
- [4] J. Foerster et al. Counterfactual Multi-Agent Policy Gradients. AAAI, 2018.
- [5] C. Finn, P. Abbeel, and S. Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. ICML, 2017.
- [6] K. Rakelly et al. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. ICML, 2019.