

**António Cruz** (140129)**, Cátia Brás** (120093)**, Ricardo Kayseller** (95813)

**Advanced Topics in Deep Learning**

MINI-PROJECT 02

DEEP REINFORCEMENT LEARNING

**DQN and PPO in LUnarLander v3**

**TAAP**

Table of Contents

[DQN & PPO Multi-Seed Training with Checkpointing & Best-Model Selection 3](#_Toc222642167)

[DQN & PPO Multi-Seed Report (Best Model Selection) 50](#_Toc222642168)

[Per-Algorithm Results 54](#_Toc222642169)

[Cross-Algorithm Comparison 59](#_Toc222642170)

[Statistical Significance 64](#_Toc222642171)

[Baseline Comparison 66](#_Toc222642172)

[Agent Behavior Analysis 69](#_Toc222642173)

[GIF Visualizations 74](#_Toc222642174)

[Appendix: Experimental Setup 76](#_Toc222642175)

[Environment Details 76](#_Toc222642176)

# DQN & PPO Multi-Seed Training with Checkpointing & Best-Model Selection

This notebook trains both DQN and PPO across multiple seeds, with: - **Periodic checkpoints** saved every N episodes - **Best-model tracking** using a combined metric (mean reward - std reward) - **Timestamped run folders** for organized model storage

Evaluates each model and produces per-algorithm, per-seed, and aggregated comparison charts.

import os, sys, random, time  
from datetime import datetime  
from typing import Callable  
  
import numpy as np  
import pandas as pd  
import torch  
import matplotlib.pyplot as plt  
  
import gymnasium as gym  
from stable\_baselines3 import DQN, PPO  
from stable\_baselines3.common.evaluation import evaluate\_policy  
from stable\_baselines3.common.vec\_env import DummyVecEnv  
from stable\_baselines3.common.callbacks import BaseCallback, EvalCallback, CallbackList  
from stable\_baselines3.common.monitor import Monitor  
  
import imageio  
from IPython.display import Image, display

# Global Configuration  
  
SEED\_LIST = [42, 123, 999]  
  
ALGORITHM\_MAP = {  
 "dqn": DQN,  
 "ppo": PPO,  
}  
  
NOTEBOOK\_DIR = os.path.dirname(os.path.abspath("\_\_file\_\_"))  
GYMNASIUM\_MODEL = "LunarLander-v3"  
MLP\_POLICY = "MlpPolicy"  
  
WIND\_ENABLED = False  
  
TOTAL\_TIMESTEPS = 1\_500\_000  
EVALUATION\_EPISODES = 20  
  
# Update live stats and plots every N episodes  
CHART\_UPDATE\_FREQ = 10  
  
# Save a training checkpoint every N episodes  
CHECKPOINT\_FREQ\_EPISODES = 100  
  
# EvalCallback: evaluate the model every N timesteps  
EVAL\_FREQ\_TIMESTEPS = 25\_000  
  
# EvalCallback: number of episodes per evaluation  
EVAL\_N\_EPISODES = 20  
  
# Solved threshold: only prefer models with mean reward >= this value  
SOLVED\_THRESHOLD = 200  
  
DEVICE = "cpu"  
  
  
def linear\_schedule(initial\_value: float) -> Callable[[float], float]:  
 """  
 Linear learning rate schedule.  
 """  
 def func(progress\_remaining: float) -> float:  
 """  
 Progress decreases from 1 (beginning) to 0 (end)  
 """  
 return progress\_remaining \* initial\_value  
 return func  
  
  
# Per-algorithm hyperparameters  
ALGO\_PARAMS = {  
 "dqn": {  
 "policy": MLP\_POLICY,  
   
 # Linear Schedule allows weights to settle perfectly at the end  
 "learning\_rate": linear\_schedule(6.3e-4),   
   
 "learning\_starts": 50\_000,  
   
 # Massive buffer prevents forgetting recovery maneuvers  
 "buffer\_size": 750\_000,   
 "batch\_size": 128,  
 "gamma": 0.99,  
   
 "exploration\_fraction": 0.12,   
   
 # High final epsilon forces the agent to keep learning recoveries  
 "exploration\_final\_eps": 0.1,   
   
 # Standard Zoo update mechanics  
 "target\_update\_interval": 250,   
 "train\_freq": 4,  
 # Takes 4 gradient updates every 4 env steps  
 "gradient\_steps": 4,  
   
 "policy\_kwargs": dict(net\_arch=[256, 256]),  
 "device": DEVICE,  
 },  
 "ppo": {  
 "learning\_rate": 2.5e-4,  
 "n\_steps": 2048,  
 "batch\_size": 64,  
 "n\_epochs": 10,  
 "gamma": 0.999,  
 "gae\_lambda": 0.95,  
 "ent\_coef": 0.01,  
 "clip\_range": 0.2,  
 },  
}  
  
print(f"Algorithms: {list(ALGORITHM\_MAP.keys())}")  
print(f"Seeds: {SEED\_LIST}")  
print(f"Wind enabled: {WIND\_ENABLED}")  
print(f"Total timesteps per seed: {TOTAL\_TIMESTEPS:,}")  
print(f"Evaluation episodes per seed: {EVALUATION\_EPISODES}")  
print(f"Chart update frequency: every {CHART\_UPDATE\_FREQ} episodes")  
print(f"Checkpoint frequency: every {CHECKPOINT\_FREQ\_EPISODES} episodes")  
print(f"Eval callback frequency: every {EVAL\_FREQ\_TIMESTEPS:,} timesteps ({EVAL\_N\_EPISODES} episodes)")  
print(f"Solved threshold: {SOLVED\_THRESHOLD}")  
print(f"Device: {DEVICE}")

Algorithms: ['dqn', 'ppo']  
Seeds: [42, 123, 999]  
Wind enabled: False  
Total timesteps per seed: 1,500,000  
Evaluation episodes per seed: 20  
Chart update frequency: every 10 episodes  
Checkpoint frequency: every 100 episodes  
Eval callback frequency: every 25,000 timesteps (20 episodes)  
Solved threshold: 200  
Device: cpu

print("Python:", sys.version.split()[0])  
print("PyTorch:", torch.\_\_version\_\_)  
print("Device:", DEVICE)  
print("CUDA:", torch.version.cuda if torch.cuda.is\_available() else "None")

Python: 3.12.3  
PyTorch: 2.10.0+cu130  
Device: cpu  
CUDA: 13.0

# Environment inspection (run once)  
env\_tmp = gym.make(GYMNASIUM\_MODEL)  
  
print("Observation space:", env\_tmp.observation\_space)  
print("Action space:", env\_tmp.action\_space)  
  
obs, info = env\_tmp.reset()  
print("Initial observation:", obs)  
  
env\_tmp.close()

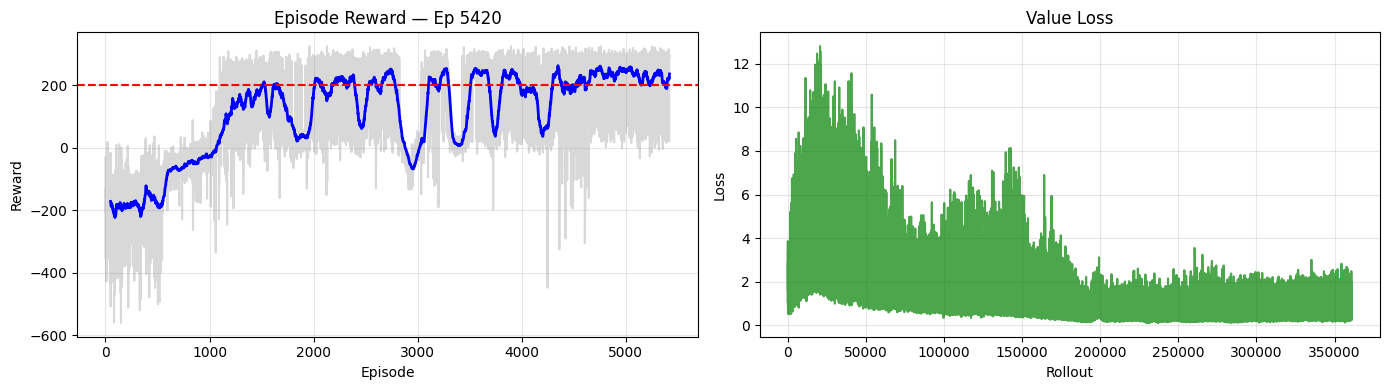
Observation space: Box([ -2.5 -2.5 -10. -10. -6.2831855 -10.  
 -0. -0. ], [ 2.5 2.5 10. 10. 6.2831855 10.  
 1. 1. ], (8,), float32)  
Action space: Discrete(4)  
Initial observation: [ 0.00419035 1.4124277 0.42442602 0.06699768 -0.00484882 -0.0961388  
 0. 0. ]

class DQNLoggingCallback(BaseCallback):  
  
 def \_\_init\_\_(self, checkpoint\_path=None, verbose: int = 0):  
 super().\_\_init\_\_(verbose)  
 self.checkpoint\_path = checkpoint\_path  
 self.episode\_rewards = []  
 self.episode\_lengths = []  
 self.value\_loss = []  
 self.entropy = []  
 self.mean\_q\_values = [] # Track Q-value overestimation  
 self.gradient\_updates = 0 # Count gradient updates  
  
 self.\_current\_reward = 0.0  
 self.\_current\_length = 0  
 self.\_plot\_handle = None  
 self.\_stats\_handle = None  
 self.\_checkpoint\_handle = None  
  
 def \_on\_step(self) -> bool:  
 rewards = self.locals.get("rewards")  
 dones = self.locals.get("dones")  
  
 if rewards is not None and dones is not None:  
 reward = rewards[0]  
 done = dones[0]  
  
 self.\_current\_reward += float(reward)  
 self.\_current\_length += 1  
  
 if done:  
 self.episode\_rewards.append(self.\_current\_reward)  
 self.episode\_lengths.append(self.\_current\_length)  
 ep = len(self.episode\_rewards)  
  
 # Periodic checkpoint  
 if self.checkpoint\_path and ep % CHECKPOINT\_FREQ\_EPISODES == 0:  
 ckpt\_path = os.path.join(self.checkpoint\_path, f"checkpoint\_ep{ep}")  
 self.model.save(ckpt\_path)  
 ckpt\_text = f"[Checkpoint] Episode {ep} saved"  
 if self.\_checkpoint\_handle is None:  
 self.\_checkpoint\_handle = display(ckpt\_text, display\_id=True)  
 else:  
 self.\_checkpoint\_handle.update(ckpt\_text)  
  
 if ep % CHART\_UPDATE\_FREQ == 0:  
 recent = np.array(self.episode\_rewards[-50:])  
 stats\_text = (  
 f'Episode {ep} | Last {len(recent)} Ep \u2014 '  
 f'Mean: {np.mean(recent):.1f} | Std: {np.std(recent):.1f} | '  
 f'Min: {np.min(recent):.1f} | Max: {np.max(recent):.1f} | '  
 f'Success: {(recent >= 200).sum() / len(recent) \* 100:.0f}%'  
 )  
 if self.\_stats\_handle is None:  
 self.\_stats\_handle = display(stats\_text, display\_id=True)  
 else:  
 self.\_stats\_handle.update(stats\_text)  
  
 if ep % CHART\_UPDATE\_FREQ == 0:  
 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))  
  
 ax1.plot(self.episode\_rewards, alpha=0.3, color='gray')  
 window = min(50, len(self.episode\_rewards))  
 rolling = pd.Series(self.episode\_rewards).rolling(window).mean()  
 ax1.plot(rolling, color='blue', linewidth=2)  
 ax1.axhline(y=200, color='red', linestyle='--')  
 ax1.set\_title(f'Episode Reward \u2014 Ep {ep}')  
 ax1.set\_xlabel('Episode')  
 ax1.set\_ylabel('Reward')  
 ax1.grid(True, alpha=0.3)  
  
 if self.value\_loss:  
 ax2.plot(self.value\_loss, color='green', alpha=0.7)  
 ax2.set\_title('Value Loss')  
 ax2.set\_xlabel('Rollout')  
 ax2.set\_ylabel('Loss')  
 ax2.grid(True, alpha=0.3)  
  
 plt.tight\_layout()  
  
 if self.\_plot\_handle is None:  
 self.\_plot\_handle = display(fig, display\_id=True)  
 else:  
 self.\_plot\_handle.update(fig)  
 plt.close(fig)  
  
 self.\_current\_reward = 0.0  
 self.\_current\_length = 0  
  
 return True  
  
 def \_on\_rollout\_end(self) -> None:  
 logger\_data = self.model.logger.name\_to\_value  
 if "train/loss" in logger\_data:  
 self.value\_loss.append(logger\_data["train/loss"])  
 if "rollout/exploration\_rate" in logger\_data:  
 self.entropy.append(logger\_data["rollout/exploration\_rate"])  
 if "train/n\_updates" in logger\_data:  
 self.gradient\_updates = int(logger\_data["train/n\_updates"])  
  
 # Sample Q-values from current observation to track overestimation  
 try:  
 obs = self.locals.get("new\_obs")  
 if obs is not None:  
 obs\_tensor = torch.as\_tensor(obs, device=self.model.device).float()  
 dqn\_model: DQN = self.model # type: ignore[assignment]  
 with torch.no\_grad():  
 q\_values = dqn\_model.q\_net(obs\_tensor)  
 self.mean\_q\_values.append(float(q\_values.max(dim=1).values.mean()))  
 except Exception:  
 pass  
  
  
class PPOLoggingCallback(BaseCallback):  
 def \_\_init\_\_(self, checkpoint\_path=None, verbose: int = 0):  
 super().\_\_init\_\_(verbose)  
 self.checkpoint\_path = checkpoint\_path  
 self.episode\_rewards = []  
 self.episode\_lengths = []  
 self.policy\_loss = []  
 self.value\_loss = []  
 self.entropy = []  
 self.clip\_fraction = [] # PPO stability: fraction of clipped updates  
 self.approx\_kl = [] # PPO stability: KL divergence  
 self.explained\_variance = [] # PPO stability: value function quality  
 self.gradient\_updates = 0 # Count gradient updates  
  
 self.\_current\_rewards: np.ndarray = np.array([])  
 self.\_current\_lengths: np.ndarray = np.array([])  
 self.\_plot\_handle = None  
 self.\_stats\_handle = None  
 self.\_checkpoint\_handle = None  
  
 def \_on\_training\_start(self) -> None:  
 n\_envs = self.training\_env.num\_envs  
 self.\_current\_rewards = np.zeros(n\_envs, dtype=np.float32)  
 self.\_current\_lengths = np.zeros(n\_envs, dtype=np.int32)  
  
 def \_on\_step(self) -> bool:  
 rewards = self.locals.get("rewards")  
 dones = self.locals.get("dones")  
  
 if rewards is not None and dones is not None:  
 self.\_current\_rewards += rewards  
 self.\_current\_lengths += 1  
  
 for i, done in enumerate(dones):  
 if done:  
 self.episode\_rewards.append(float(self.\_current\_rewards[i]))  
 self.episode\_lengths.append(int(self.\_current\_lengths[i]))  
 ep = len(self.episode\_rewards)  
  
 # Periodic checkpoint  
 if self.checkpoint\_path and ep % CHECKPOINT\_FREQ\_EPISODES == 0:  
 ckpt\_path = os.path.join(self.checkpoint\_path, f"checkpoint\_ep{ep}")  
 self.model.save(ckpt\_path)  
 ckpt\_text = f"[Checkpoint] Episode {ep} saved"  
 if self.\_checkpoint\_handle is None:  
 self.\_checkpoint\_handle = display(ckpt\_text, display\_id=True)  
 else:  
 self.\_checkpoint\_handle.update(ckpt\_text)  
  
 if ep % CHART\_UPDATE\_FREQ == 0:  
 recent = np.array(self.episode\_rewards[-50:])  
 stats\_text = (  
 f'Episode {ep} | Last {len(recent)} Ep \u2014 '  
 f'Mean: {np.mean(recent):.1f} | Std: {np.std(recent):.1f} | '  
 f'Min: {np.min(recent):.1f} | Max: {np.max(recent):.1f} | '  
 f'Success: {(recent >= 200).sum() / len(recent) \* 100:.0f}%'  
 )  
 if self.\_stats\_handle is None:  
 self.\_stats\_handle = display(stats\_text, display\_id=True)  
 else:  
 self.\_stats\_handle.update(stats\_text)  
  
 if ep % CHART\_UPDATE\_FREQ == 0:  
 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))  
  
 ax1.plot(self.episode\_rewards, alpha=0.3, color='gray')  
 window = min(50, len(self.episode\_rewards))  
 rolling = pd.Series(self.episode\_rewards).rolling(window).mean()  
 ax1.plot(rolling, color='blue', linewidth=2)  
 ax1.axhline(y=200, color='red', linestyle='--')  
 ax1.set\_title(f'Episode Reward \u2014 Ep {ep}')  
 ax1.set\_xlabel('Episode')  
 ax1.set\_ylabel('Reward')  
 ax1.grid(True, alpha=0.3)  
  
 if self.value\_loss:  
 ax2.plot(self.value\_loss, color='green', alpha=0.7)  
 ax2.set\_title('Value Loss')  
 ax2.set\_xlabel('Rollout')  
 ax2.set\_ylabel('Loss')  
 ax2.grid(True, alpha=0.3)  
  
 plt.tight\_layout()  
  
 if self.\_plot\_handle is None:  
 self.\_plot\_handle = display(fig, display\_id=True)  
 else:  
 self.\_plot\_handle.update(fig)  
 plt.close(fig)  
  
 self.\_current\_rewards[i] = 0  
 self.\_current\_lengths[i] = 0  
 return True  
  
 def \_on\_rollout\_end(self) -> None:  
 logger\_data = self.model.logger.name\_to\_value  
 if "train/policy\_gradient\_loss" in logger\_data:  
 self.policy\_loss.append(logger\_data["train/policy\_gradient\_loss"])  
 if "train/value\_loss" in logger\_data:  
 self.value\_loss.append(logger\_data["train/value\_loss"])  
 if "train/entropy\_loss" in logger\_data:  
 self.entropy.append(-logger\_data["train/entropy\_loss"])  
 if "train/clip\_fraction" in logger\_data:  
 self.clip\_fraction.append(logger\_data["train/clip\_fraction"])  
 if "train/approx\_kl" in logger\_data:  
 self.approx\_kl.append(logger\_data["train/approx\_kl"])  
 if "train/explained\_variance" in logger\_data:  
 self.explained\_variance.append(logger\_data["train/explained\_variance"])  
 if "train/n\_updates" in logger\_data:  
 self.gradient\_updates = int(logger\_data["train/n\_updates"])  
  
  
class CombinedMetricEvalCallback(EvalCallback):  
 """  
 Custom EvalCallback that selects the best model using a combined metric:  
 score = mean\_reward - std\_reward  
 This favors models that are both high-performing and consistent.  
  
 Two-tier selection with solved gate:  
 - Once any evaluation has mean\_reward >= SOLVED\_THRESHOLD, only solved  
 models can replace the current best (unsolved fallbacks are discarded).  
 - Before any model solves, the overall best score is tracked as fallback.  
 """  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 self.best\_combined\_score = -np.inf  
 self.best\_std\_reward = np.inf  
 self.best\_success\_rate = 0.0  
 self.best\_timestep = 0  
 self.\_any\_solved = False  
 self.\_eval\_handle = None  
  
 def \_on\_step(self) -> bool:  
 continue\_training = True  
  
 if self.eval\_freq > 0 and self.n\_calls % self.eval\_freq == 0:  
 episode\_rewards, episode\_lengths = evaluate\_policy(  
 self.model,  
 self.eval\_env,  
 n\_eval\_episodes=self.n\_eval\_episodes,  
 render=self.render,  
 deterministic=self.deterministic,  
 return\_episode\_rewards=True,  
 )  
  
 mean\_reward, std\_reward = np.mean(episode\_rewards), np.std(episode\_rewards)  
 mean\_ep\_length = np.mean(episode\_lengths)  
 success\_rate = np.sum(np.array(episode\_rewards) >= 200) / len(episode\_rewards) \* 100 # type: ignore  
 self.last\_mean\_reward = mean\_reward  
  
 combined\_score = mean\_reward - std\_reward  
 is\_solved = mean\_reward >= SOLVED\_THRESHOLD  
  
 # Two-tier best model selection:  
 # 1. If this model is solved, save if it's the first solved or has better score  
 # 2. If no model has solved yet, save the overall best as fallback  
 save\_new\_best = False  
 if is\_solved:  
 if not self.\_any\_solved:  
 # First solved model — always save (replaces any unsolved fallback)  
 save\_new\_best = True  
 self.\_any\_solved = True  
 elif combined\_score > self.best\_combined\_score:  
 # Better solved model  
 save\_new\_best = True  
 elif not self.\_any\_solved:  
 # No solved model yet — track overall best as fallback  
 if combined\_score > self.best\_combined\_score:  
 save\_new\_best = True  
  
 solved\_tag = " [SOLVED]" if is\_solved else ""  
 eval\_text = (  
 f"Eval @ {self.num\_timesteps} steps | "  
 f"Reward: {mean\_reward:.2f} +/- {std\_reward:.2f}{solved\_tag} | "  
 f"Success: {success\_rate:.0f}% | "  
 f"Score (mean-std): {combined\_score:.2f} | "  
 f"Best: {self.best\_combined\_score:.2f}"  
 )  
  
 if save\_new\_best:  
 eval\_text += f" >> New best model!"  
 self.best\_combined\_score = combined\_score  
 self.best\_mean\_reward = mean\_reward  
 self.best\_std\_reward = std\_reward  
 self.best\_success\_rate = success\_rate  
 self.best\_timestep = self.num\_timesteps  
 if self.best\_model\_save\_path is not None:  
 self.model.save(  
 os.path.join(self.best\_model\_save\_path, "best\_model")  
 )  
  
 if self.\_eval\_handle is None:  
 self.\_eval\_handle = display(eval\_text, display\_id=True)  
 else:  
 self.\_eval\_handle.update(eval\_text)  
  
 if self.log\_path is not None:  
 self.evaluations\_timesteps.append(self.num\_timesteps)  
 self.evaluations\_results.append(episode\_rewards) # type: ignore  
 self.evaluations\_length.append(episode\_lengths) # type: ignore  
 np.savez(  
 self.log\_path,  
 timesteps=self.evaluations\_timesteps,  
 results=self.evaluations\_results,  
 ep\_lengths=self.evaluations\_length,  
 )  
  
 self.logger.record("eval/mean\_reward", float(mean\_reward))  
 self.logger.record("eval/std\_reward", float(std\_reward))  
 self.logger.record("eval/mean\_ep\_length", float(mean\_ep\_length))  
 self.logger.record("eval/combined\_score", float(combined\_score))  
 self.logger.record("eval/success\_rate", float(success\_rate))  
  
 return continue\_training  
  
  
CALLBACK\_MAP = {  
 "dqn": DQNLoggingCallback,  
 "ppo": PPOLoggingCallback,  
}

def set\_all\_seeds(seed):  
 random.seed(seed)  
 np.random.seed(seed)  
 torch.manual\_seed(seed)  
 torch.use\_deterministic\_algorithms(True)  
 if torch.cuda.is\_available():  
 torch.cuda.manual\_seed(seed)  
 torch.cuda.manual\_seed\_all(seed)  
 torch.backends.cudnn.deterministic = True  
 torch.backends.cudnn.benchmark = False

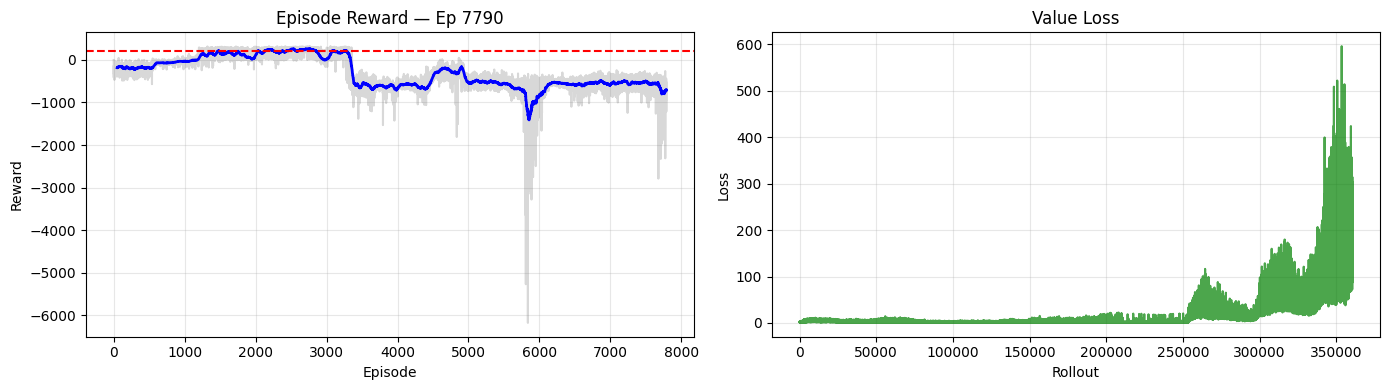
# Training loop: algorithms x seeds  
  
training\_results = {} # {algo: {seed: callback}}  
training\_times = {} # {algo: {seed: seconds}}  
model\_save\_paths = {} # {algo: {seed: {"run\_dir", "final", "best"}}}  
eval\_callbacks = {} # {algo: {seed: eval\_cb}} — for best-model summary table  
  
for algo\_name, algo\_class in ALGORITHM\_MAP.items():  
  
 tb\_dir = os.path.join(NOTEBOOK\_DIR, "outputs\_" + algo\_name, "tensorboard")  
  
 training\_results[algo\_name] = {}  
 training\_times[algo\_name] = {}  
 model\_save\_paths[algo\_name] = {}  
 eval\_callbacks[algo\_name] = {}  
  
 for seed in SEED\_LIST:  
 print(f"\n{'='\*60}")  
 print(f"{algo\_name.upper()} | Seed {seed}")  
 print(f"{'='\*60}\n")  
  
 # Create timestamped run directory  
 run\_timestamp = datetime.now().strftime("%Y-%m-%d\_%H\_%M\_%S")  
 run\_dir = os.path.join(NOTEBOOK\_DIR, "../../../models", algo\_name, run\_timestamp)  
 checkpoints\_dir = os.path.join(run\_dir, "checkpoints")  
 os.makedirs(checkpoints\_dir, exist\_ok=True)  
  
 set\_all\_seeds(seed)  
  
 def make\_env(s=seed):  
 env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 env.reset(seed=s)  
 return env  
  
 env = DummyVecEnv([make\_env])  
 env.seed(seed)  
  
 # Separate eval env for EvalCallback  
 def make\_eval\_env(s=seed):  
 e = Monitor(gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED))  
 e.reset(seed=s)  
 return e  
  
 eval\_env = DummyVecEnv([make\_eval\_env])  
  
 params = {  
 "policy": MLP\_POLICY,  
 "env": env,  
 "device": DEVICE,  
 "seed": seed,  
 "tensorboard\_log": tb\_dir,  
 \*\*ALGO\_PARAMS[algo\_name],  
 }  
  
 model = algo\_class(\*\*params)  
  
 # Setup callbacks  
 logging\_cb = CALLBACK\_MAP[algo\_name](checkpoint\_path=checkpoints\_dir)  
 eval\_cb = CombinedMetricEvalCallback(  
 eval\_env=eval\_env,  
 eval\_freq=EVAL\_FREQ\_TIMESTEPS,  
 n\_eval\_episodes=EVAL\_N\_EPISODES,  
 best\_model\_save\_path=run\_dir,  
 log\_path=os.path.join(run\_dir, "eval\_log"),  
 deterministic=True,  
 verbose=1,  
 )  
  
 t\_start = time.time()  
 model.learn(  
 total\_timesteps=TOTAL\_TIMESTEPS,  
 callback=CallbackList([logging\_cb, eval\_cb]),  
 progress\_bar=True,  
 )  
 t\_elapsed = time.time() - t\_start  
  
 training\_times[algo\_name][seed] = t\_elapsed  
 print(f"\nTraining time: {t\_elapsed/60:.1f} min ({t\_elapsed:.0f} s)")  
  
 # Save final model  
 final\_path = os.path.join(run\_dir, f"lab009\_{algo\_name}\_{seed}")  
 model.save(final\_path)  
 print(f"Final model: {final\_path}")  
 print(f"Best model: {os.path.join(run\_dir, 'best\_model')}")  
 print(f"Checkpoints: {checkpoints\_dir}")  
 print(  
 f"Best model stats: "  
 f"Reward: {eval\_cb.best\_mean\_reward:.2f} +/- {eval\_cb.best\_std\_reward:.2f} | "  
 f"Success: {eval\_cb.best\_success\_rate:.0f}% | "  
 f"Score (mean-std): {eval\_cb.best\_combined\_score:.2f} | "  
 f"@ {eval\_cb.best\_timestep:,} steps"  
 )  
  
 model\_save\_paths[algo\_name][seed] = {  
 "run\_dir": run\_dir,  
 "final": final\_path,  
 "best": os.path.join(run\_dir, "best\_model"),  
 }  
  
 training\_results[algo\_name][seed] = logging\_cb  
 eval\_callbacks[algo\_name][seed] = eval\_cb  
  
 env.close()  
 eval\_env.close()  
  
 print(f"\n{algo\_name.upper()}: All {len(SEED\_LIST)} seeds trained.")  
  
# Best Model Summary Table  
print(f"\n{'='\*60}")  
print("BEST MODEL SUMMARY (all algorithms x seeds)")  
print(f"{'='\*60}\n")  
  
best\_rows = []  
for algo\_name in ALGORITHM\_MAP:  
 for seed in SEED\_LIST:  
 cb = eval\_callbacks[algo\_name][seed]  
 best\_rows.append({  
 "Algorithm": algo\_name.upper(),  
 "Seed": seed,  
 "Mean Reward": f"{cb.best\_mean\_reward:.2f}",  
 "Std Reward": f"{cb.best\_std\_reward:.2f}",  
 "Success": f"{cb.best\_success\_rate:.0f}%",  
 "Score (mean-std)": f"{cb.best\_combined\_score:.2f}",  
 "@ Timestep": f"{cb.best\_timestep:,}",  
 })  
  
print(pd.DataFrame(best\_rows).to\_string(index=False))  
print(f"\nAll training complete.")

============================================================  
DQN | Seed 42  
============================================================  
  
  
  
  
Output()  
  
  
  
'Episode 5420 | Last 50 Ep — Mean: 236.2 | Std: 95.5 | Min: 15.8 | Max: 315.9 | Success: 82%'



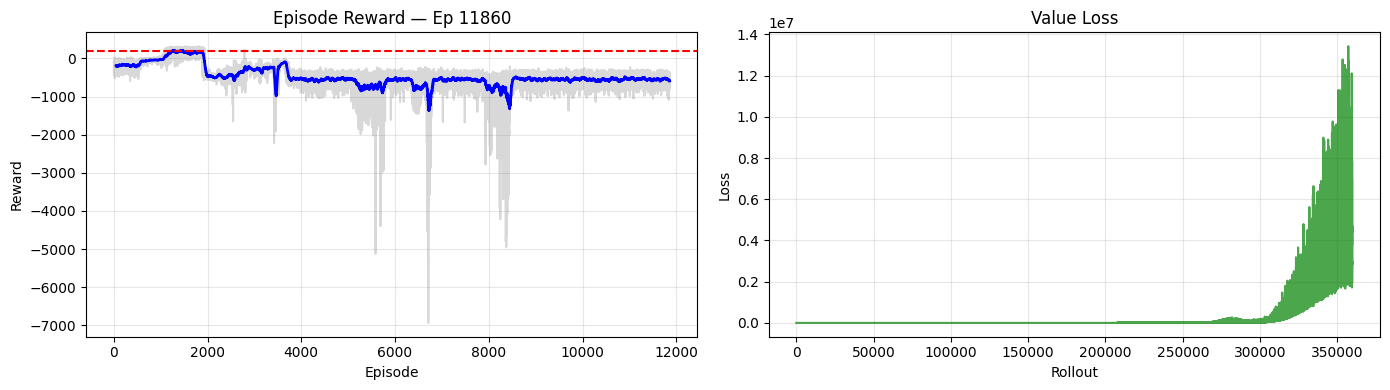
'[Checkpoint] Episode 5400 saved'  
  
  
  
'Eval @ 1500000 steps | Reward: 276.50 +/- 16.66 [SOLVED] | Success: 100% | Score (mean-std): 259.85 | Best: 266.35'

Output()  
  
  
  
'Episode 7790 | Last 50 Ep — Mean: -708.5 | Std: 349.2 | Min: -2311.8 | Max: -262.3 | Success: 0%'



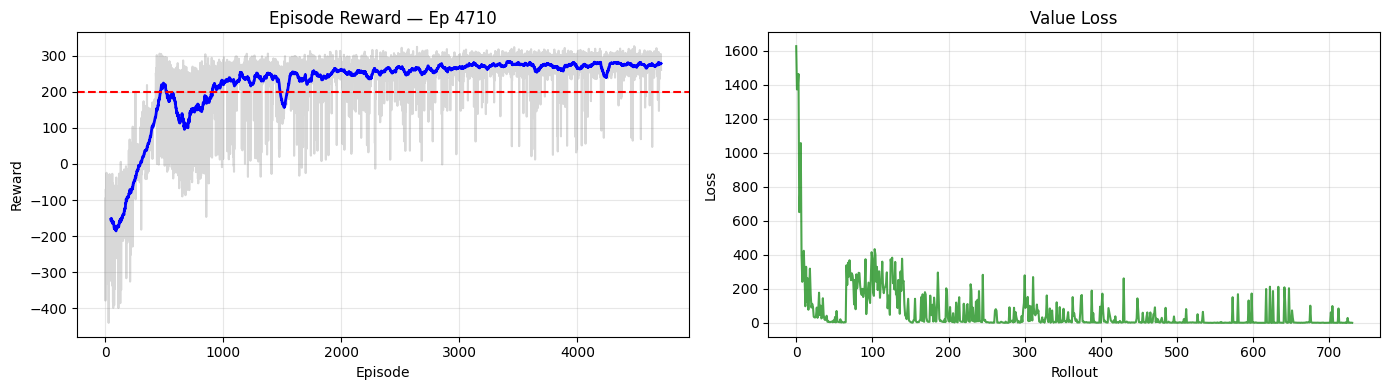
Training time: 52.4 min (3146 s)  
Final model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_20\_43\_20/lab009\_dqn\_42  
Best model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_20\_43\_20/best\_model  
Checkpoints: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_20\_43\_20/checkpoints  
Best model stats: Reward: 283.74 +/- 17.39 | Success: 100% | Score (mean-std): 266.35 | @ 1,425,000 steps  
  
============================================================  
DQN | Seed 123  
============================================================  
  
  
  
  
'[Checkpoint] Episode 7700 saved'  
  
  
  
'Eval @ 1500000 steps | Reward: -926.29 +/- 368.67 | Success: 0% | Score (mean-std): -1294.96 | Best: 259.82'

Output()  
  
  
  
'Episode 11860 | Last 50 Ep — Mean: -590.1 | Std: 162.5 | Min: -1082.2 | Max: -348.5 | Success: 0%'



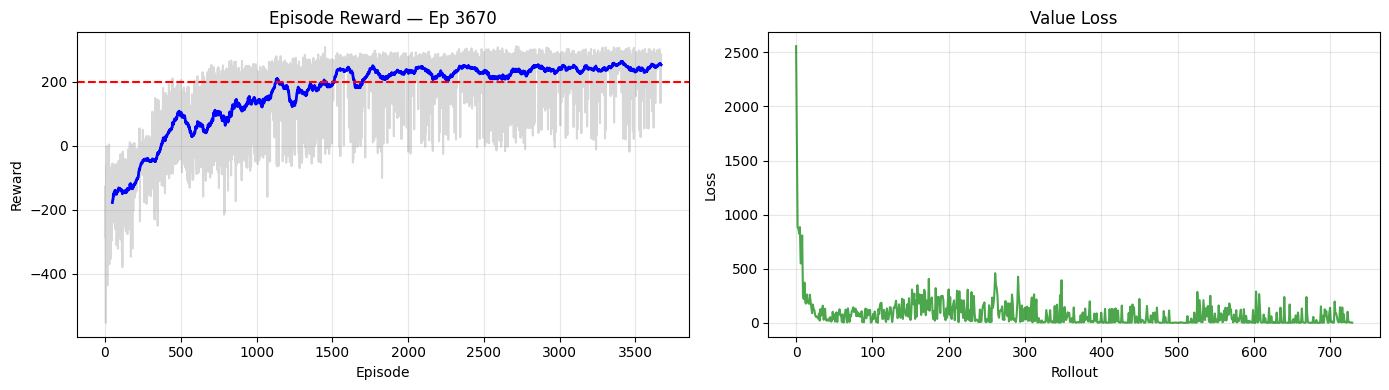
Training time: 51.5 min (3093 s)  
Final model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_21\_35\_46/lab009\_dqn\_123  
Best model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_21\_35\_46/best\_model  
Checkpoints: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_21\_35\_46/checkpoints  
Best model stats: Reward: 275.61 +/- 15.80 | Success: 100% | Score (mean-std): 259.82 | @ 875,000 steps  
  
============================================================  
DQN | Seed 999  
============================================================  
  
  
  
  
'[Checkpoint] Episode 11800 saved'  
  
  
  
'Eval @ 1500000 steps | Reward: -593.75 +/- 172.38 | Success: 0% | Score (mean-std): -766.14 | Best: 243.87'

Output()  
  
  
  
Training time: 50.4 min (3025 s)  
Final model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_22\_27\_19/lab009\_dqn\_999  
Best model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_22\_27\_19/best\_model  
Checkpoints: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_22\_27\_19/checkpoints  
Best model stats: Reward: 266.25 +/- 22.38 | Success: 100% | Score (mean-std): 243.87 | @ 550,000 steps  
  
DQN: All 3 seeds trained.  
  
============================================================  
PPO | Seed 42  
============================================================  
  
  
  
  
'Episode 4710 | Last 50 Ep — Mean: 278.3 | Std: 31.3 | Min: 146.7 | Max: 320.5 | Success: 96%'



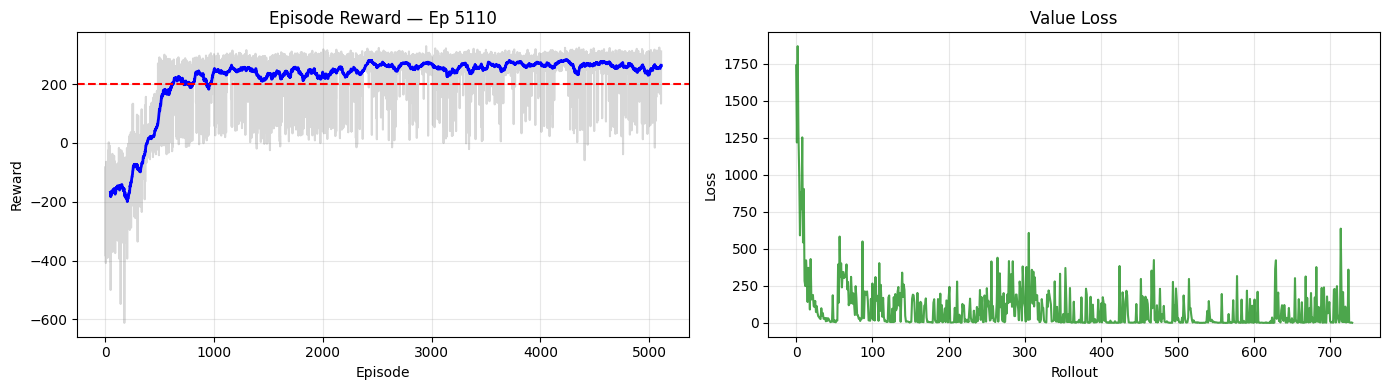
'[Checkpoint] Episode 4700 saved'  
  
  
  
'Eval @ 1500000 steps | Reward: 289.50 +/- 21.48 [SOLVED] | Success: 100% | Score (mean-std): 268.02 | Best: 266.36 >> New best model!'

Output()  
  
  
  
Training time: 21.6 min (1294 s)  
Final model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_17\_45/lab009\_ppo\_42  
Best model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_17\_45/best\_model  
Checkpoints: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_17\_45/checkpoints  
Best model stats: Reward: 289.50 +/- 21.48 | Success: 100% | Score (mean-std): 268.02 | @ 1,500,000 steps  
  
============================================================  
PPO | Seed 123  
============================================================  
  
  
  
  
'Episode 3670 | Last 50 Ep — Mean: 252.4 | Std: 46.9 | Min: 56.7 | Max: 301.1 | Success: 90%'



'[Checkpoint] Episode 3600 saved'  
  
  
  
'Eval @ 1500000 steps | Reward: 257.63 +/- 35.30 [SOLVED] | Success: 95% | Score (mean-std): 222.33 | Best: 249.47'

Output()  
  
  
  
Training time: 24.6 min (1475 s)  
Final model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_39\_19/lab009\_ppo\_123  
Best model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_39\_19/best\_model  
Checkpoints: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_39\_19/checkpoints  
Best model stats: Reward: 267.61 +/- 18.14 | Success: 100% | Score (mean-std): 249.47 | @ 1,300,000 steps  
  
============================================================  
PPO | Seed 999  
============================================================  
  
  
  
  
'Episode 5110 | Last 50 Ep — Mean: 264.0 | Std: 47.3 | Min: 83.3 | Max: 323.9 | Success: 90%'



'[Checkpoint] Episode 5100 saved'  
  
  
  
'Eval @ 1500000 steps | Reward: 271.94 +/- 35.27 [SOLVED] | Success: 95% | Score (mean-std): 236.66 | Best: 266.00'

Training time: 20.6 min (1235 s)  
Final model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-22\_00\_03\_53/lab009\_ppo\_999  
Best model: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-22\_00\_03\_53/best\_model  
Checkpoints: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-22\_00\_03\_53/checkpoints  
Best model stats: Reward: 285.68 +/- 19.69 | Success: 100% | Score (mean-std): 266.00 | @ 1,175,000 steps  
  
PPO: All 3 seeds trained.  
  
============================================================  
BEST MODEL SUMMARY (all algorithms x seeds)  
============================================================  
  
Algorithm Seed Mean Reward Std Reward Success Score (mean-std) @ Timestep  
 DQN 42 283.74 17.39 100% 266.35 1,425,000  
 DQN 123 275.61 15.80 100% 259.82 875,000  
 DQN 999 266.25 22.38 100% 243.87 550,000  
 PPO 42 289.50 21.48 100% 268.02 1,500,000  
 PPO 123 267.61 18.14 100% 249.47 1,300,000  
 PPO 999 285.68 19.69 100% 266.00 1,175,000  
  
All training complete.

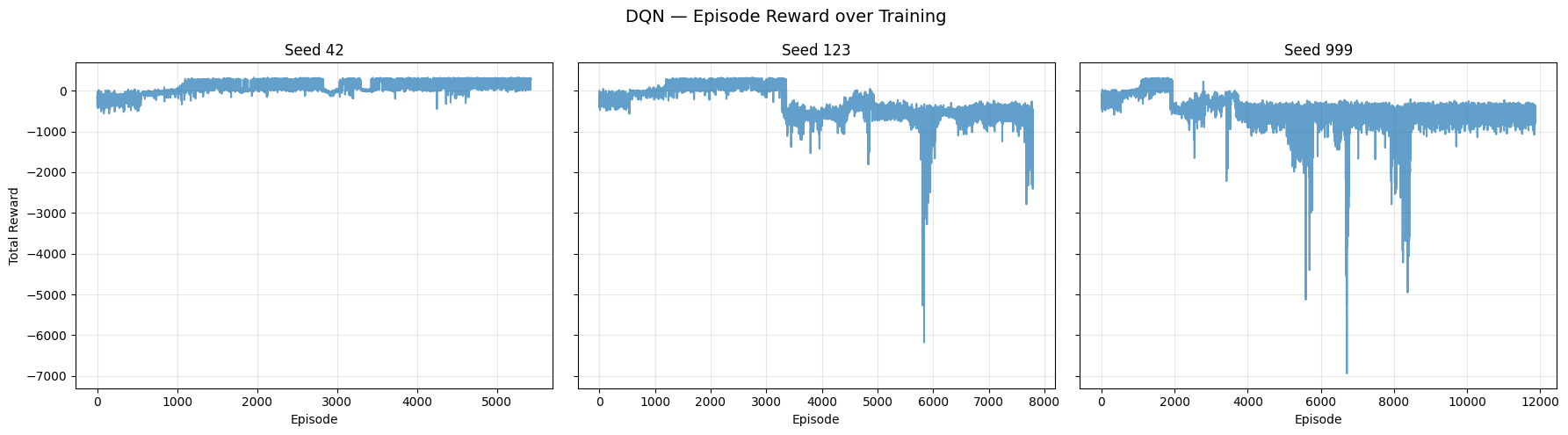
# Training Time Summary  
  
rows = []  
for algo\_name in ALGORITHM\_MAP:  
 for seed in SEED\_LIST:  
 t = training\_times[algo\_name][seed]  
 rows.append({  
 "Algorithm": algo\_name.upper(),  
 "Seed": seed,  
 "Time (s)": f"{t:.0f}",  
 "Time (min)": f"{t/60:.1f}",  
 })  
  
for algo\_name in ALGORITHM\_MAP:  
 times = list(training\_times[algo\_name].values())  
 rows.append({  
 "Algorithm": algo\_name.upper(),  
 "Seed": "Mean",  
 "Time (s)": f"{np.mean(times):.0f}",  
 "Time (min)": f"{np.mean(times)/60:.1f}",  
 })  
  
print("\*\*\* TRAINING TIME SUMMARY \*\*\*")  
print(f"Timesteps per seed: {TOTAL\_TIMESTEPS:,} | Device: {DEVICE}")  
print()  
print(pd.DataFrame(rows).to\_string(index=False))

\*\*\* TRAINING TIME SUMMARY \*\*\*  
Timesteps per seed: 1,500,000 | Device: cpu  
  
Algorithm Seed Time (s) Time (min)  
 DQN 42 3146 52.4  
 DQN 123 3093 51.5  
 DQN 999 3025 50.4  
 PPO 42 1294 21.6  
 PPO 123 1475 24.6  
 PPO 999 1235 20.6  
 DQN Mean 3088 51.5  
 PPO Mean 1335 22.2

# Gradient Updates Summary  
# DQN: train\_freq=4, gradient\_steps=4 -> 4 gradient steps every 4 env steps  
# PPO: n\_steps=2048, n\_epochs=10, batch\_size=128 -> 10 \* (2048/128) = 160 updates per rollout  
  
rows = []  
for algo\_name in ALGORITHM\_MAP:  
 # Use the actual count from the last seed's callback  
 last\_seed = SEED\_LIST[-1]  
 actual\_updates = training\_results[algo\_name][last\_seed].gradient\_updates  
  
 # Also compute analytically for verification  
 if algo\_name == "dqn":  
 p = ALGO\_PARAMS["dqn"]  
 train\_freq = p.get("train\_freq", 4)  
 grad\_steps = p.get("gradient\_steps", -1)  
 learning\_starts = p.get("learning\_starts", 0)  
 effective\_steps = TOTAL\_TIMESTEPS - learning\_starts  
 steps\_per\_call = train\_freq if grad\_steps == -1 else grad\_steps  
 analytical = (effective\_steps // train\_freq) \* steps\_per\_call  
 else:  
 p = ALGO\_PARAMS["ppo"]  
 n\_steps = p.get("n\_steps", 2048)  
 n\_epochs = p.get("n\_epochs", 10)  
 batch\_size = p.get("batch\_size", 128)  
 n\_rollouts = TOTAL\_TIMESTEPS // n\_steps  
 minibatches\_per\_epoch = n\_steps // batch\_size  
 analytical = n\_rollouts \* n\_epochs \* minibatches\_per\_epoch  
  
 rows.append({  
 "Algorithm": algo\_name.upper(),  
 "Actual (from training)": f"{actual\_updates:,}",  
 "Analytical (computed)": f"{analytical:,}",  
 "Total Env Steps": f"{TOTAL\_TIMESTEPS:,}",  
 "Ratio (updates/steps)": f"{actual\_updates / TOTAL\_TIMESTEPS:.3f}",  
 })  
  
print("\*\*\* GRADIENT UPDATES SUMMARY \*\*\*")  
print(pd.DataFrame(rows).to\_string(index=False))  
print()  
print("DQN: train\_freq=4, gradient\_steps=4 -> 4 gradient updates per training call, called every 4 env steps")  
print(f"PPO: n\_steps={ALGO\_PARAMS['ppo']['n\_steps']}, n\_epochs={ALGO\_PARAMS['ppo']['n\_epochs']}, "  
 f"batch\_size={ALGO\_PARAMS['ppo']['batch\_size']} -> "  
 f"{ALGO\_PARAMS['ppo']['n\_epochs'] \* (ALGO\_PARAMS['ppo']['n\_steps'] // ALGO\_PARAMS['ppo']['batch\_size'])} "  
 f"updates per rollout")

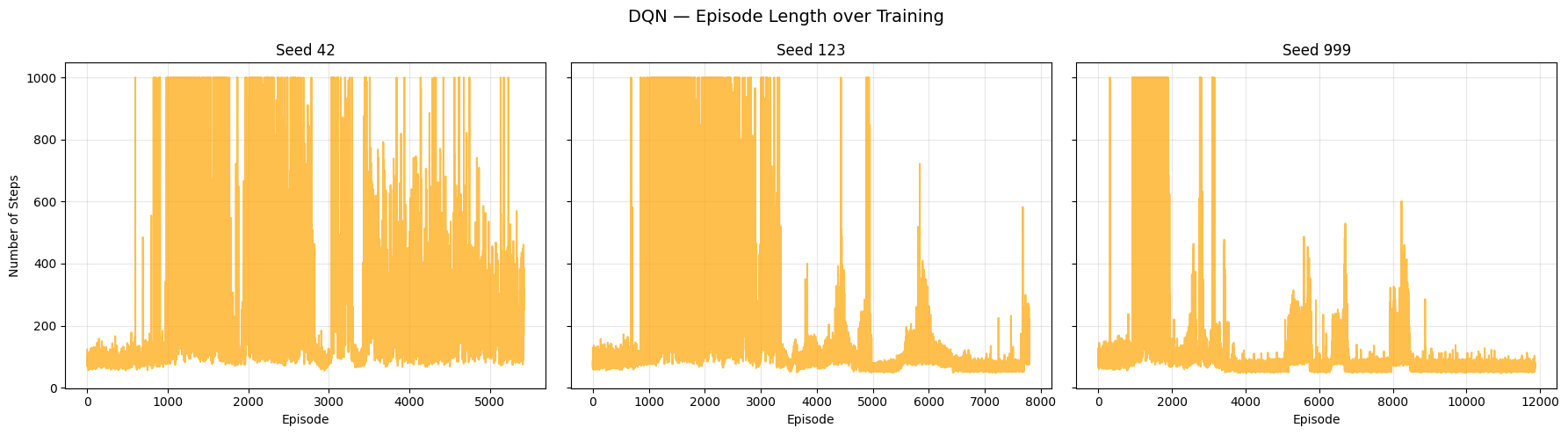
\*\*\* GRADIENT UPDATES SUMMARY \*\*\*  
Algorithm Actual (from training) Analytical (computed) Total Env Steps Ratio (updates/steps)  
 DQN 1,449,996 1,450,000 1,500,000 0.967  
 PPO 7,320 234,240 1,500,000 0.005  
  
DQN: train\_freq=4, gradient\_steps=4 -> 4 gradient updates per training call, called every 4 env steps  
PPO: n\_steps=2048, n\_epochs=10, batch\_size=64 -> 320 updates per rollout

# Per-Algorithm, Per-Seed: Episode Reward over Training  
  
for algo\_name in ALGORITHM\_MAP:  
 fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
 if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
 for ax, seed in zip(axes, SEED\_LIST):  
 ax.plot(training\_results[algo\_name][seed].episode\_rewards, alpha=0.7)  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Episode")  
 ax.grid(True, alpha=0.3)  
  
 axes[0].set\_ylabel("Total Reward")  
 fig.suptitle(f"{algo\_name.upper()} \u2014 Episode Reward over Training", fontsize=14)  
 plt.tight\_layout()  
 plt.show()



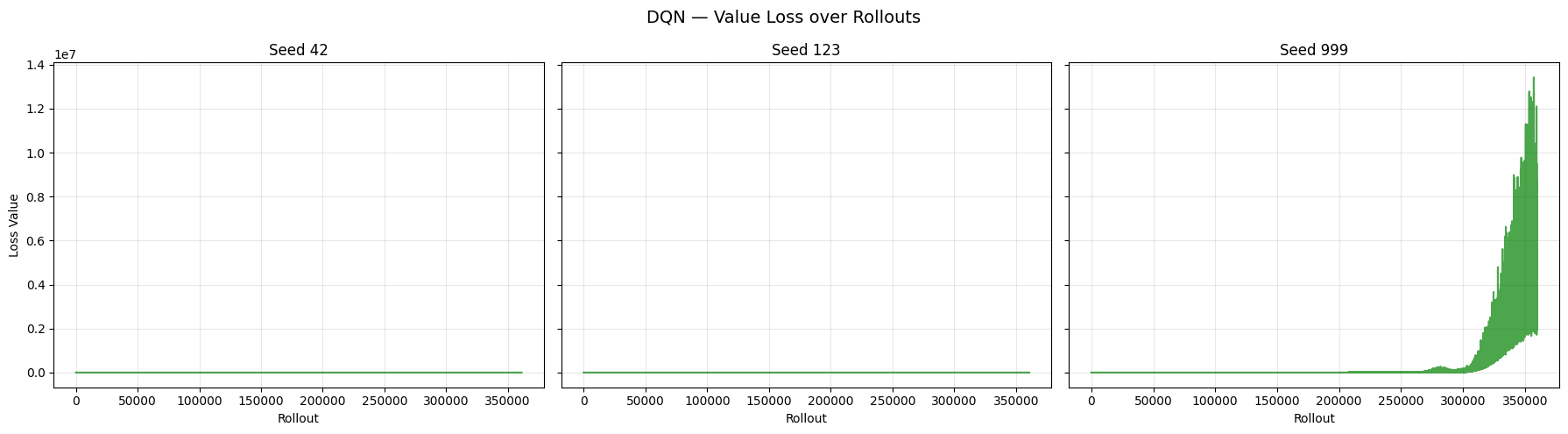


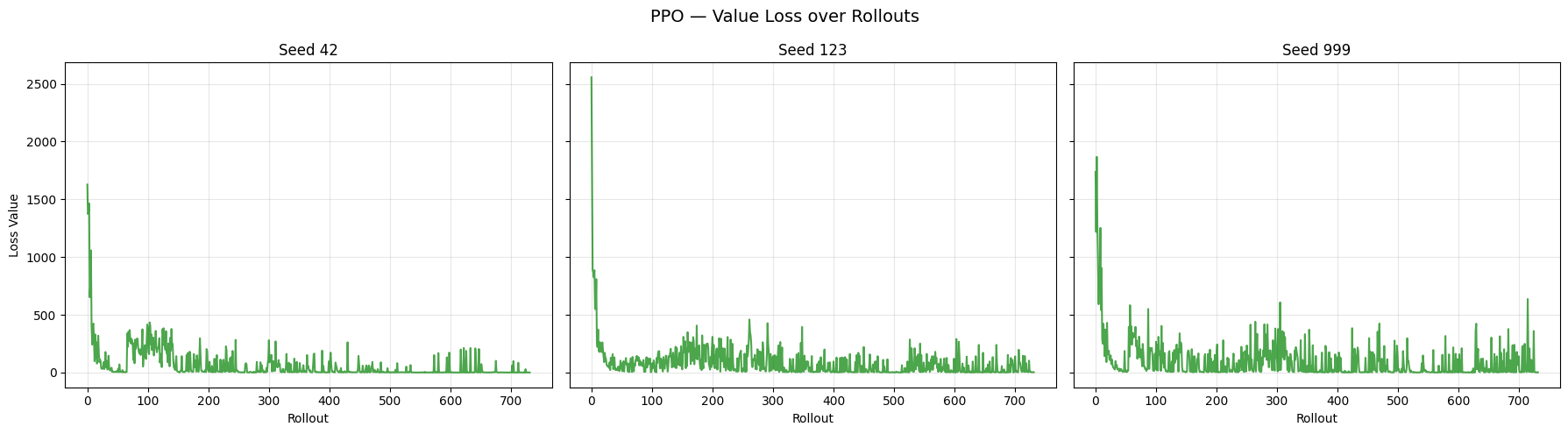
# Per-Algorithm, Per-Seed: Episode Length over Training  
  
for algo\_name in ALGORITHM\_MAP:  
 fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
 if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
 for ax, seed in zip(axes, SEED\_LIST):  
 ax.plot(training\_results[algo\_name][seed].episode\_lengths, alpha=0.7, color="orange")  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Episode")  
 ax.grid(True, alpha=0.3)  
  
 axes[0].set\_ylabel("Number of Steps")  
 fig.suptitle(f"{algo\_name.upper()} \u2014 Episode Length over Training", fontsize=14)  
 plt.tight\_layout()  
 plt.show()



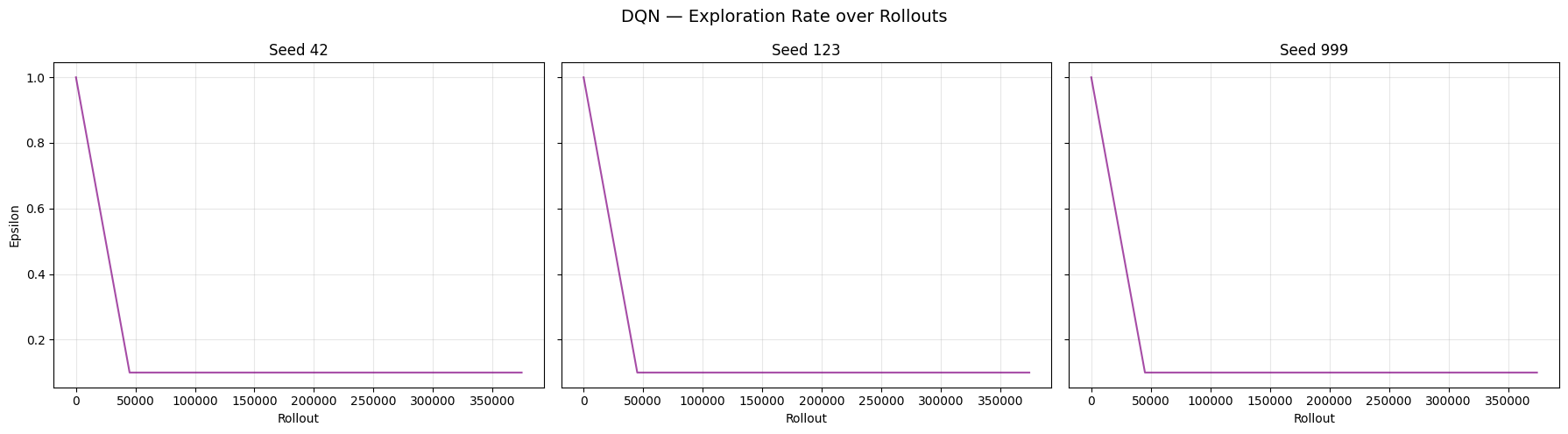


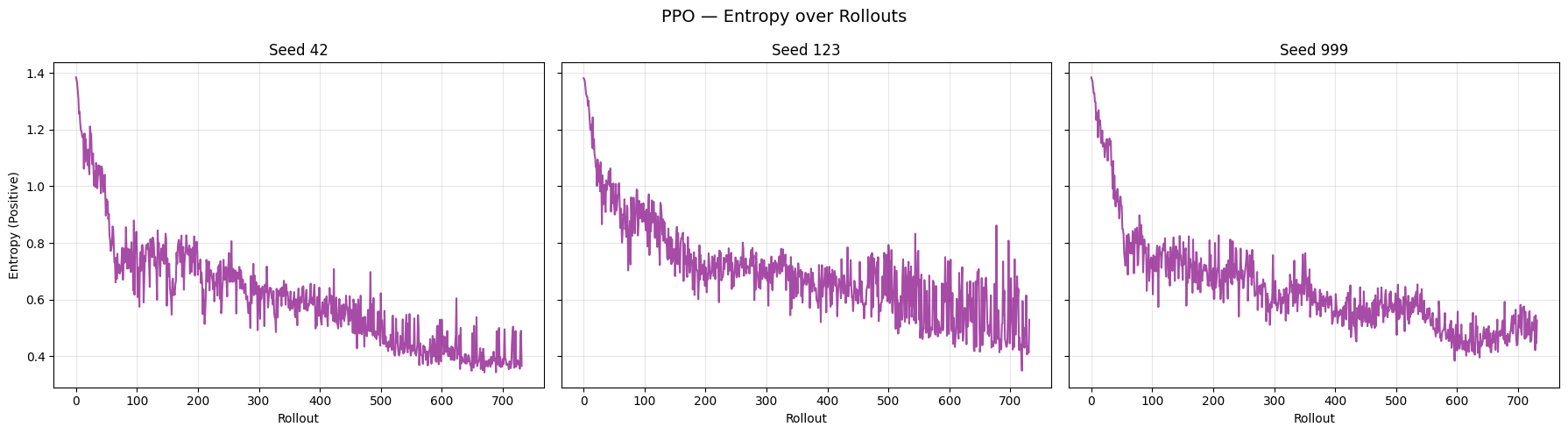
# Per-Algorithm, Per-Seed: Value Loss over Rollouts  
  
for algo\_name in ALGORITHM\_MAP:  
 fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
 if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
 for ax, seed in zip(axes, SEED\_LIST):  
 ax.plot(training\_results[algo\_name][seed].value\_loss, alpha=0.7, color="green")  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Rollout")  
 ax.grid(True, alpha=0.3)  
  
 axes[0].set\_ylabel("Loss Value")  
 fig.suptitle(f"{algo\_name.upper()} \u2014 Value Loss over Rollouts", fontsize=14)  
 plt.tight\_layout()  
 plt.show()



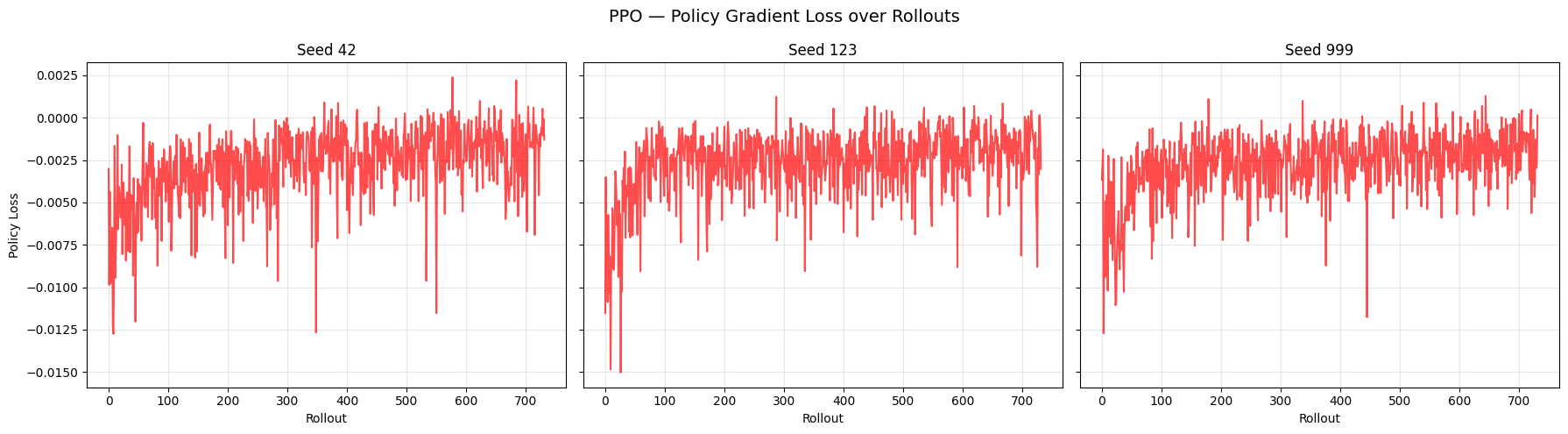


# Per-Algorithm, Per-Seed: Entropy / Exploration Rate over Rollouts  
  
entropy\_labels = {"dqn": ("Epsilon", "Exploration Rate"), "ppo": ("Entropy (Positive)", "Entropy")}  
  
for algo\_name in ALGORITHM\_MAP:  
 ylabel, title\_suffix = entropy\_labels[algo\_name]  
  
 fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
 if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
 for ax, seed in zip(axes, SEED\_LIST):  
 ax.plot(training\_results[algo\_name][seed].entropy, alpha=0.7, color="purple")  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Rollout")  
 ax.grid(True, alpha=0.3)  
  
 axes[0].set\_ylabel(ylabel)  
 fig.suptitle(f"{algo\_name.upper()} \u2014 {title\_suffix} over Rollouts", fontsize=14)  
 plt.tight\_layout()  
 plt.show()

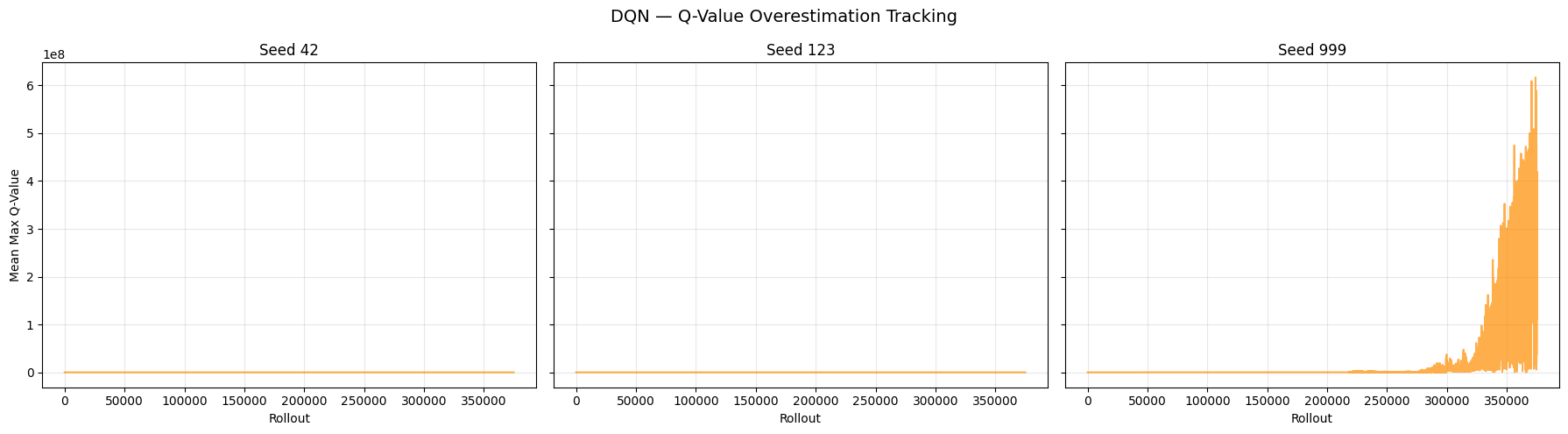




# PPO Only: Policy Loss over Rollouts  
  
fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
for ax, seed in zip(axes, SEED\_LIST):  
 ax.plot(training\_results["ppo"][seed].policy\_loss, alpha=0.7, color="red")  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Rollout")  
 ax.grid(True, alpha=0.3)  
  
axes[0].set\_ylabel("Policy Loss")  
fig.suptitle("PPO \u2014 Policy Gradient Loss over Rollouts", fontsize=14)  
plt.tight\_layout()  
plt.show()

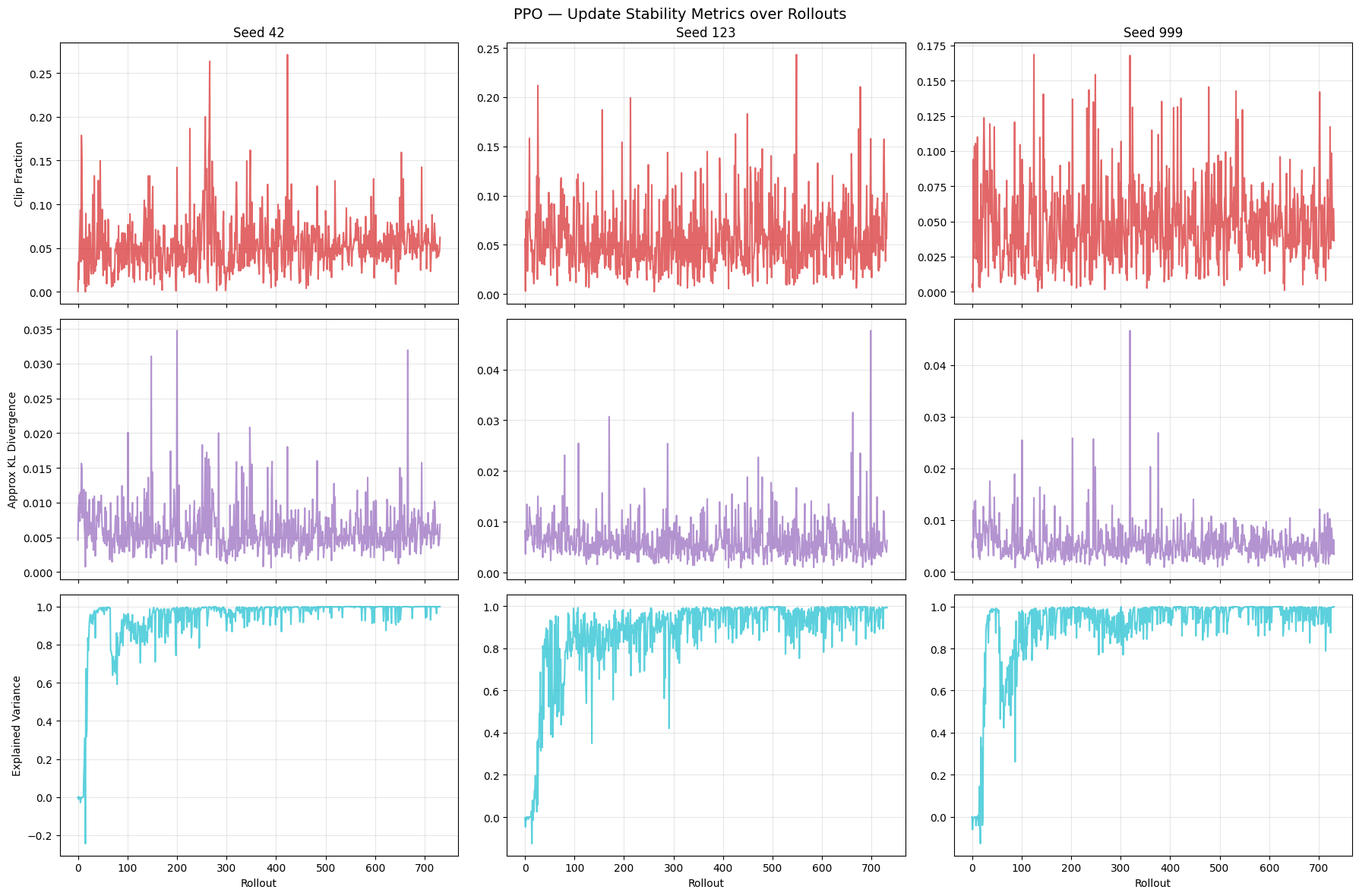


# DQN Overestimation: Mean Max Q-Value over Training  
  
fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
for ax, seed in zip(axes, SEED\_LIST):  
 q\_vals = training\_results["dqn"][seed].mean\_q\_values  
 if q\_vals:  
 ax.plot(q\_vals, alpha=0.7, color="darkorange")  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Rollout")  
 ax.grid(True, alpha=0.3)  
  
axes[0].set\_ylabel("Mean Max Q-Value")  
fig.suptitle("DQN \u2014 Q-Value Overestimation Tracking", fontsize=14)  
plt.tight\_layout()  
plt.show()  
  
print("Note: Steadily rising Q-values that diverge from actual returns indicate overestimation.")  
print("A stable or slowly growing curve suggests the target network is controlling overestimation.")



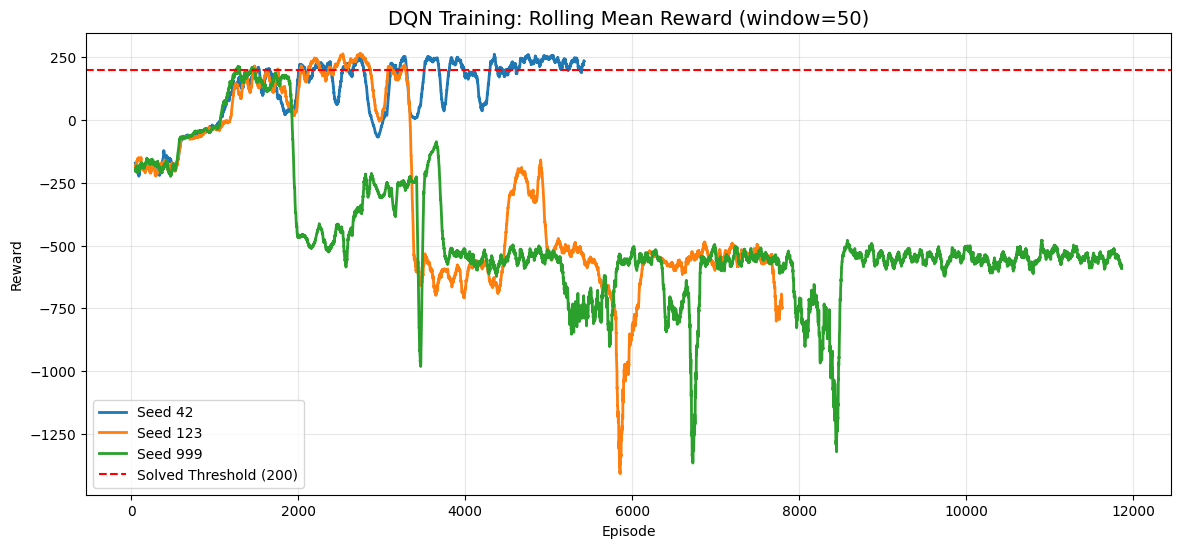
Note: Steadily rising Q-values that diverge from actual returns indicate overestimation.  
A stable or slowly growing curve suggests the target network is controlling overestimation.

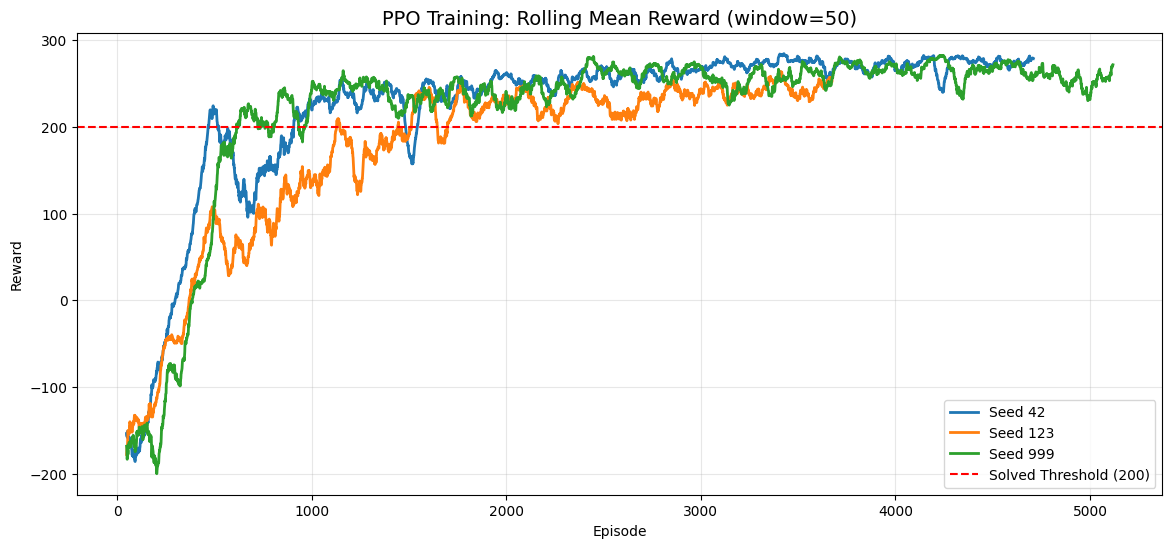
# PPO Update Stability: Clip Fraction, Approx KL, Explained Variance  
  
fig, axes = plt.subplots(3, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 12), sharex=True)  
if len(SEED\_LIST) == 1:  
 axes = axes.reshape(3, 1)  
  
metrics = [  
 ("clip\_fraction", "Clip Fraction", "tab:red",  
 "Fraction of policy updates clipped by PPO. High values suggest the policy is changing too fast."),  
 ("approx\_kl", "Approx KL Divergence", "tab:purple",  
 "KL divergence between old and new policy. Spikes indicate large policy shifts."),  
 ("explained\_variance", "Explained Variance", "tab:cyan",  
 "How well the value function predicts returns. 1.0 = perfect, 0 = no better than mean."),  
]  
  
for row, (attr, ylabel, color, \_) in enumerate(metrics):  
 for col, seed in enumerate(SEED\_LIST):  
 data = getattr(training\_results["ppo"][seed], attr)  
 if data:  
 axes[row][col].plot(data, alpha=0.7, color=color)  
 if row == 0:  
 axes[row][col].set\_title(f"Seed {seed}")  
 if row == 2:  
 axes[row][col].set\_xlabel("Rollout")  
 axes[row][col].grid(True, alpha=0.3)  
 axes[row][0].set\_ylabel(ylabel)  
  
fig.suptitle("PPO \u2014 Update Stability Metrics over Rollouts", fontsize=14)  
plt.tight\_layout()  
plt.show()  
  
for \_, ylabel, \_, description in metrics:  
 print(f" {ylabel}: {description}")



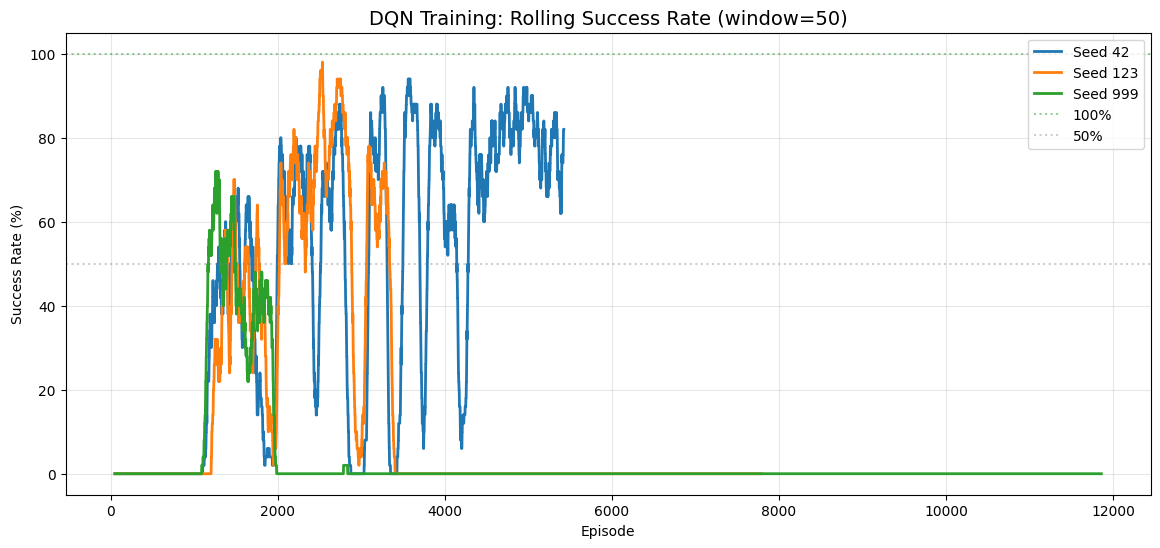
Clip Fraction: Fraction of policy updates clipped by PPO. High values suggest the policy is changing too fast.  
 Approx KL Divergence: KL divergence between old and new policy. Spikes indicate large policy shifts.  
 Explained Variance: How well the value function predicts returns. 1.0 = perfect, 0 = no better than mean.

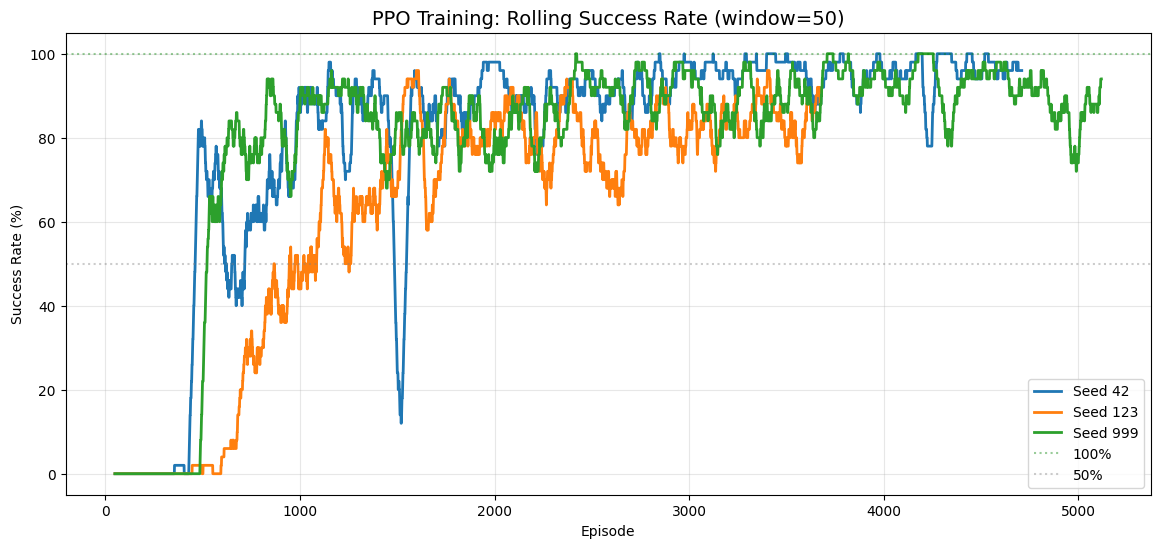
# Aggregated: Rolling Reward Overlay — per algorithm (all seeds on one chart)  
  
seed\_colors = list(plt.colormaps["tab10"](range(10))) # type: ignore[arg-type]  
  
for algo\_name in ALGORITHM\_MAP:  
 plt.figure(figsize=(14, 6))  
 for i, seed in enumerate(SEED\_LIST):  
 rewards = training\_results[algo\_name][seed].episode\_rewards  
 rolling = pd.Series(rewards).rolling(50).mean()  
 plt.plot(rolling, color=seed\_colors[i], linewidth=2, label=f"Seed {seed}")  
  
 plt.axhline(y=200, color='red', linestyle='--', label='Solved Threshold (200)')  
 plt.title(f"{algo\_name.upper()} Training: Rolling Mean Reward (window=50)", fontsize=14)  
 plt.xlabel("Episode")  
 plt.ylabel("Reward")  
 plt.legend()  
 plt.grid(True, alpha=0.3)  
 plt.show()



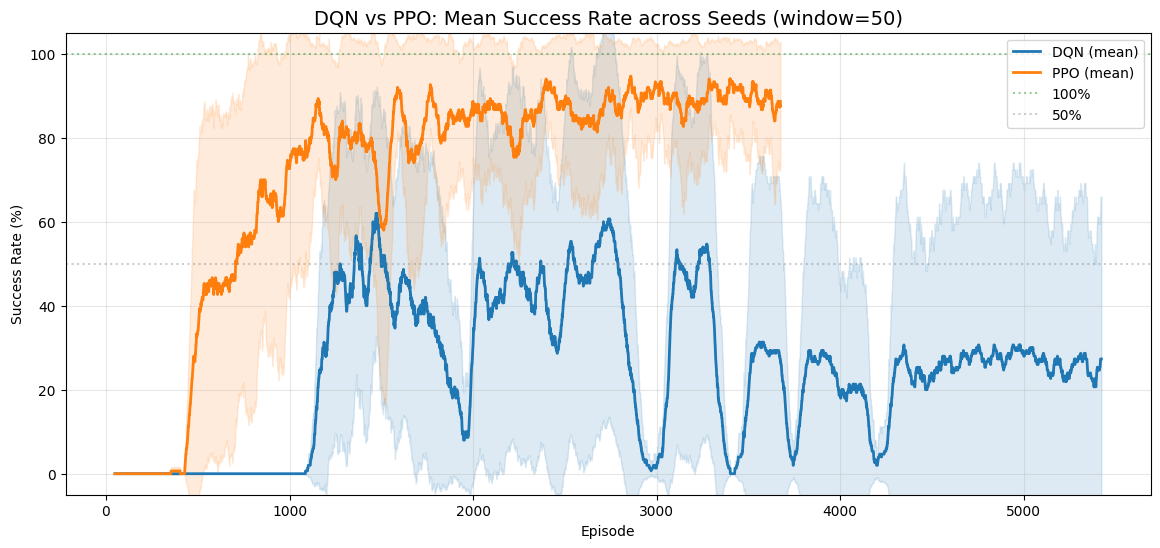


# Per-Algorithm: Rolling Success Rate over Training (window=50)  
  
for algo\_name in ALGORITHM\_MAP:  
 plt.figure(figsize=(14, 6))  
 for i, seed in enumerate(SEED\_LIST):  
 rewards = np.array(training\_results[algo\_name][seed].episode\_rewards)  
 success = (rewards >= 200).astype(float)  
 rolling\_success = pd.Series(success).rolling(50).mean() \* 100  
 plt.plot(rolling\_success, color=seed\_colors[i], linewidth=2, label=f"Seed {seed}")  
  
 plt.axhline(y=100, color='green', linestyle=':', alpha=0.4, label='100%')  
 plt.axhline(y=50, color='gray', linestyle=':', alpha=0.4, label='50%')  
 plt.title(f"{algo\_name.upper()} Training: Rolling Success Rate (window=50)", fontsize=14)  
 plt.xlabel("Episode")  
 plt.ylabel("Success Rate (%)")  
 plt.ylim(-5, 105)  
 plt.legend()  
 plt.grid(True, alpha=0.3)  
 plt.show()

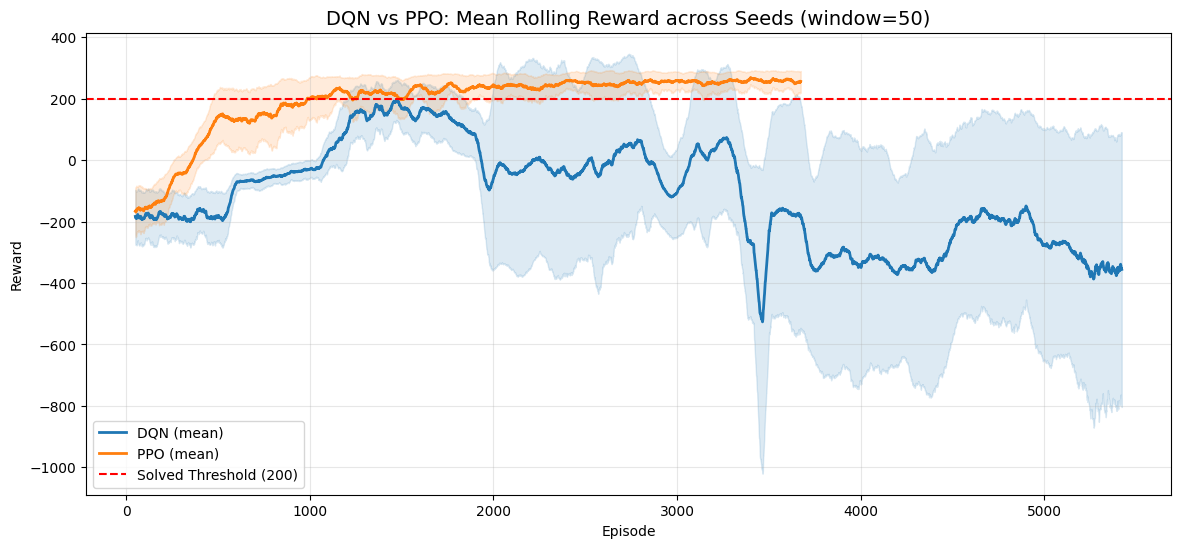




# Cross-Algorithm Comparison: Success Rate over Training (averaged across seeds)  
  
algo\_colors = {"dqn": "tab:blue", "ppo": "tab:orange"}  
  
plt.figure(figsize=(14, 6))  
for algo\_name in ALGORITHM\_MAP:  
 min\_len = min(len(training\_results[algo\_name][s].episode\_rewards) for s in SEED\_LIST)  
 all\_rewards = np.array([training\_results[algo\_name][s].episode\_rewards[:min\_len] for s in SEED\_LIST])  
 all\_success = (all\_rewards >= 200).astype(float)  
  
 mean\_success = pd.Series(all\_success.mean(axis=0)).rolling(50).mean() \* 100  
 std\_success = pd.Series(all\_success.std(axis=0)).rolling(50).mean() \* 100  
  
 episodes = np.arange(len(mean\_success))  
 plt.plot(episodes, mean\_success, color=algo\_colors[algo\_name], linewidth=2,  
 label=f"{algo\_name.upper()} (mean)")  
 plt.fill\_between(episodes, mean\_success - std\_success, mean\_success + std\_success,  
 color=algo\_colors[algo\_name], alpha=0.15)  
  
plt.axhline(y=100, color='green', linestyle=':', alpha=0.4, label='100%')  
plt.axhline(y=50, color='gray', linestyle=':', alpha=0.4, label='50%')  
plt.title("DQN vs PPO: Mean Success Rate across Seeds (window=50)", fontsize=14)  
plt.xlabel("Episode")  
plt.ylabel("Success Rate (%)")  
plt.ylim(-5, 105)  
plt.legend()  
plt.grid(True, alpha=0.3)  
plt.show()



# Cross-Algorithm Comparison: Rolling Reward (averaged across seeds)  
  
algo\_colors = {"dqn": "tab:blue", "ppo": "tab:orange"}  
  
plt.figure(figsize=(14, 6))  
for algo\_name in ALGORITHM\_MAP:  
 # Find the shortest episode count across seeds for alignment  
 min\_len = min(len(training\_results[algo\_name][s].episode\_rewards) for s in SEED\_LIST)  
 all\_rewards = np.array([training\_results[algo\_name][s].episode\_rewards[:min\_len] for s in SEED\_LIST])  
 mean\_rewards = pd.Series(all\_rewards.mean(axis=0)).rolling(50).mean()  
 std\_rewards = pd.Series(all\_rewards.std(axis=0)).rolling(50).mean()  
  
 episodes = np.arange(len(mean\_rewards))  
 plt.plot(episodes, mean\_rewards, color=algo\_colors[algo\_name], linewidth=2, label=f"{algo\_name.upper()} (mean)")  
 plt.fill\_between(episodes, mean\_rewards - std\_rewards, mean\_rewards + std\_rewards,  
 color=algo\_colors[algo\_name], alpha=0.15)  
  
plt.axhline(y=200, color='red', linestyle='--', label='Solved Threshold (200)')  
plt.title("DQN vs PPO: Mean Rolling Reward across Seeds (window=50)", fontsize=14)  
plt.xlabel("Episode")  
plt.ylabel("Reward")  
plt.legend()  
plt.grid(True, alpha=0.3)  
plt.show()



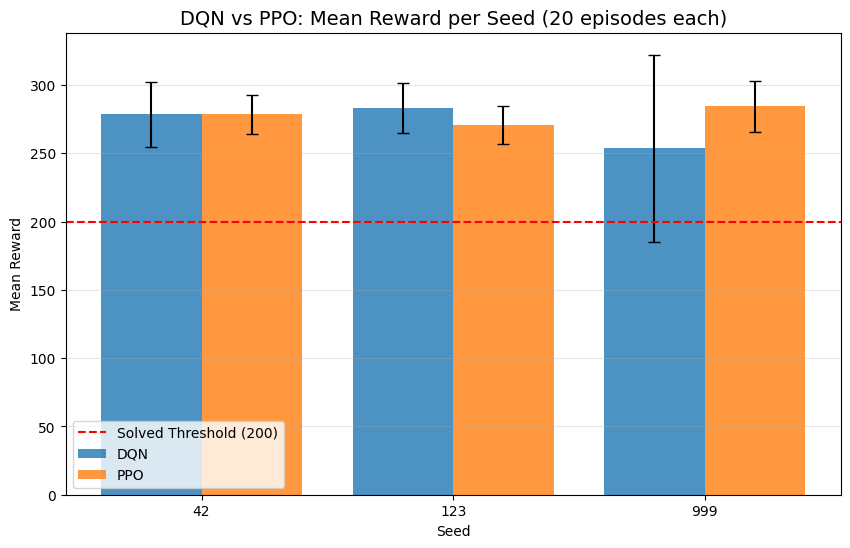
# Evaluation: deterministic episodes per algorithm per seed (best model)  
  
evaluation\_results = {} # {algo: {seed: np.array}}  
  
for algo\_name, algo\_class in ALGORITHM\_MAP.items():  
 evaluation\_results[algo\_name] = {}  
  
 for seed in SEED\_LIST:  
 print(f"Evaluating {algo\_name.upper()} seed {seed} (best model)...")  
  
 set\_all\_seeds(seed)  
  
 best\_path = model\_save\_paths[algo\_name][seed]["best"]  
  
 def make\_eval\_env(s=seed):  
 env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 env.reset(seed=s)  
 return env  
  
 eval\_model = algo\_class.load(best\_path, env=DummyVecEnv([make\_eval\_env]), device=DEVICE)  
  
 eval\_env = Monitor(gym.make(GYMNASIUM\_MODEL, enable\_wind=WIND\_ENABLED))  
 eval\_env.reset(seed=seed)  
  
 rewards, \_ = evaluate\_policy(  
 eval\_model,  
 eval\_env,  
 n\_eval\_episodes=EVALUATION\_EPISODES,  
 deterministic=True,  
 return\_episode\_rewards=True  
 )  
  
 evaluation\_results[algo\_name][seed] = np.array(rewards)  
 eval\_env.close()  
  
print(f"\nEvaluation complete for all algorithms and seeds.")

Evaluating DQN seed 42 (best model)...  
Evaluating DQN seed 123 (best model)...  
Evaluating DQN seed 999 (best model)...  
Evaluating PPO seed 42 (best model)...  
Evaluating PPO seed 123 (best model)...  
Evaluating PPO seed 999 (best model)...  
  
Evaluation complete for all algorithms and seeds.

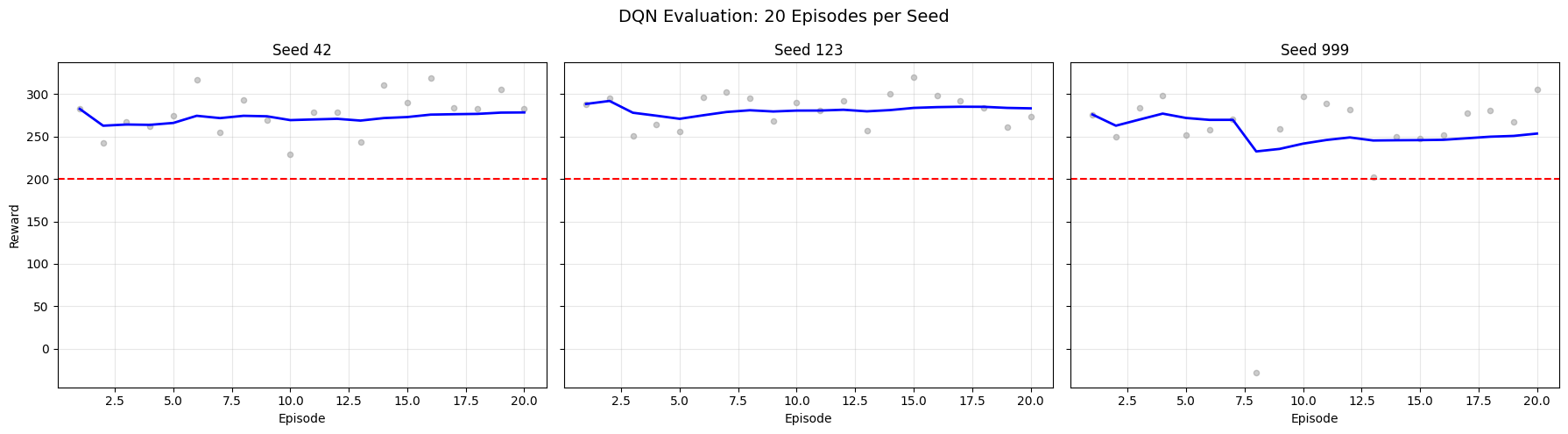
# Evaluation Summary Tables (per algorithm + overall)  
  
for algo\_name in ALGORITHM\_MAP:  
 rows = []  
 for seed in SEED\_LIST:  
 r = evaluation\_results[algo\_name][seed]  
 rows.append({  
 "Seed": seed,  
 "Mean Reward": f"{np.mean(r):.2f}",  
 "Std Dev": f"{np.std(r):.2f}",  
 "Min Reward": f"{np.min(r):.2f}",  
 "Max Reward": f"{np.max(r):.2f}",  
 "Success Rate": f"{(r >= 200).sum() / len(r) \* 100:.1f}%"  
 })  
  
 all\_r = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
 rows.append({  
 "Seed": "Overall",  
 "Mean Reward": f"{np.mean(all\_r):.2f}",  
 "Std Dev": f"{np.std(all\_r):.2f}",  
 "Min Reward": f"{np.min(all\_r):.2f}",  
 "Max Reward": f"{np.max(all\_r):.2f}",  
 "Success Rate": f"{(all\_r >= 200).sum() / len(all\_r) \* 100:.1f}%"  
 })  
  
 print(f"\*\*\* {algo\_name.upper()} MULTI-SEED EVALUATION SUMMARY \*\*\*")  
 print(f"Episodes per seed: {EVALUATION\_EPISODES} | Total: {len(all\_r)}")  
 print(pd.DataFrame(rows).to\_string(index=False))  
 print()

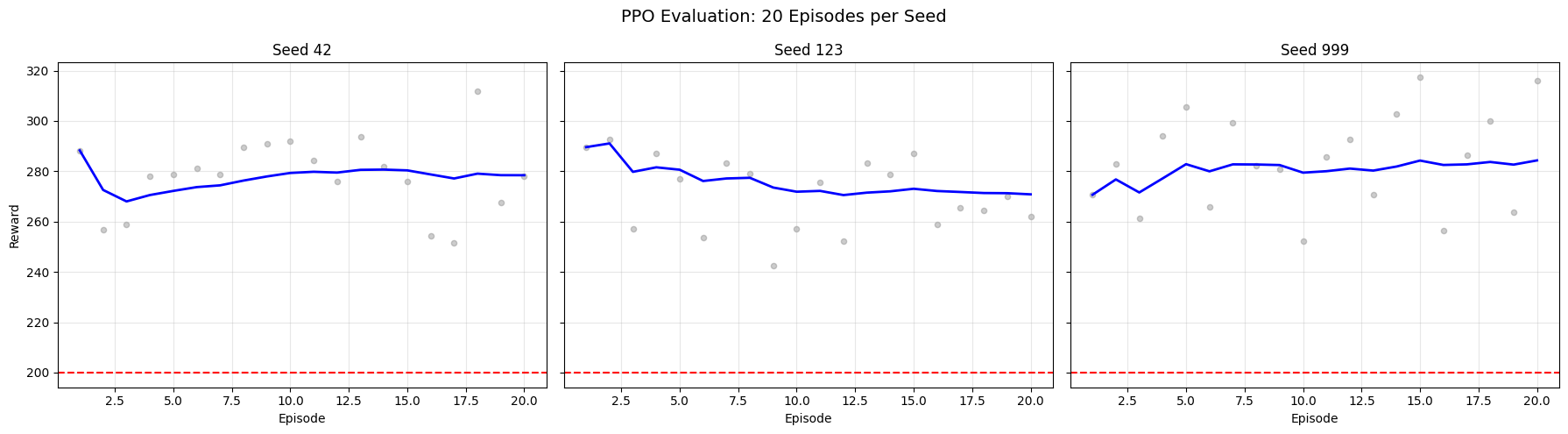
\*\*\* DQN MULTI-SEED EVALUATION SUMMARY \*\*\*  
Episodes per seed: 20 | Total: 60  
 Seed Mean Reward Std Dev Min Reward Max Reward Success Rate  
 42 278.51 23.70 228.67 318.97 100.0%  
 123 283.34 18.19 250.33 319.93 100.0%  
 999 253.57 68.63 -28.26 305.91 95.0%  
Overall 271.80 45.14 -28.26 319.93 98.3%  
  
\*\*\* PPO MULTI-SEED EVALUATION SUMMARY \*\*\*  
Episodes per seed: 20 | Total: 60  
 Seed Mean Reward Std Dev Min Reward Max Reward Success Rate  
 42 278.43 14.56 251.45 311.70 100.0%  
 123 270.83 14.06 242.53 292.55 100.0%  
 999 284.32 18.77 252.25 317.36 100.0%  
Overall 277.86 16.87 242.53 317.36 100.0%

# Cross-Algorithm Comparison: Bar Chart  
  
algo\_names = list(ALGORITHM\_MAP.keys())  
n\_algos = len(algo\_names)  
n\_seeds = len(SEED\_LIST)  
bar\_width = 0.8 / n\_algos  
x = np.arange(n\_seeds)  
  
plt.figure(figsize=(max(10, 3 \* n\_seeds), 6))  
for i, algo\_name in enumerate(algo\_names):  
 means = [np.mean(evaluation\_results[algo\_name][s]) for s in SEED\_LIST]  
 stds = [np.std(evaluation\_results[algo\_name][s]) for s in SEED\_LIST]  
 offset = (i - (n\_algos - 1) / 2) \* bar\_width  
 plt.bar(x + offset, means, bar\_width, yerr=stds, capsize=4,  
 label=algo\_name.upper(), alpha=0.8)  
  
plt.axhline(y=200, color='red', linestyle='--', label='Solved Threshold (200)')  
plt.xticks(x, [str(s) for s in SEED\_LIST])  
plt.title(f"DQN vs PPO: Mean Reward per Seed ({EVALUATION\_EPISODES} episodes each)", fontsize=14)  
plt.xlabel("Seed")  
plt.ylabel("Mean Reward")  
plt.legend()  
plt.grid(True, alpha=0.3, axis='y')  
plt.show()



# Per-Algorithm: Evaluation Convergence Plots  
  
for algo\_name in ALGORITHM\_MAP:  
 fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
 if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
 for ax, seed in zip(axes, SEED\_LIST):  
 rewards = evaluation\_results[algo\_name][seed]  
 episodes = np.arange(1, len(rewards) + 1)  
 running\_mean = np.cumsum(rewards) / episodes  
  
 ax.scatter(episodes, rewards, color='gray', alpha=0.4, s=20)  
 ax.plot(episodes, running\_mean, color='blue', linewidth=2)  
 ax.axhline(y=200, color='red', linestyle='--')  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Episode")  
 ax.grid(True, alpha=0.3)  
  
 axes[0].set\_ylabel("Reward")  
 fig.suptitle(f"{algo\_name.upper()} Evaluation: {EVALUATION\_EPISODES} Episodes per Seed", fontsize=14)  
 plt.tight\_layout()  
 plt.show()





# GIF Visualization (one per algorithm per seed, best model)  
  
for algo\_name, algo\_class in ALGORITHM\_MAP.items():  
 output\_dir = os.path.join(NOTEBOOK\_DIR, "outputs\_" + algo\_name)  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 for seed in SEED\_LIST:  
 print(f"Generating GIF for {algo\_name.upper()} seed {seed} (best model)...")  
  
 best\_path = model\_save\_paths[algo\_name][seed]["best"]  
  
 def make\_vis\_env(s=seed):  
 env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 env.reset(seed=s)  
 return env  
  
 vis\_model = algo\_class.load(best\_path, env=DummyVecEnv([make\_vis\_env]), device=DEVICE)  
  
 vis\_env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 frames = []  
 obs, info = vis\_env.reset(seed=seed)  
 done = False  
  
 while not done:  
 action, \_ = vis\_model.predict(obs, deterministic=True)  
 obs, reward, terminated, truncated, info = vis\_env.step(action)  
 done = terminated or truncated  
 frames.append(vis\_env.render())  
  
 vis\_env.close()  
  
 gif\_path = os.path.join(output\_dir, f"{algo\_name}\_seed{seed}.gif")  
 imageio.mimsave(gif\_path, frames, fps=30)  
 print(f" Saved: {gif\_path}")  
 display(Image(filename=gif\_path))

Generating GIF for DQN seed 42 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_dqn/dqn\_seed42.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for DQN seed 123 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_dqn/dqn\_seed123.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for DQN seed 999 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_dqn/dqn\_seed999.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for PPO seed 42 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_ppo/ppo\_seed42.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for PPO seed 123 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_ppo/ppo\_seed123.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for PPO seed 999 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_ppo/ppo\_seed999.gif  
  
  
  
<IPython.core.display.Image object>

# Hyperparameter Tables  
  
for algo\_name in ALGORITHM\_MAP:  
 params = ALGO\_PARAMS[algo\_name]  
 rows = [{"Parameter": k, "Value": str(v)} for k, v in params.items()]  
 rows.append({"Parameter": "total\_timesteps", "Value": str(TOTAL\_TIMESTEPS)})  
 rows.append({"Parameter": "device", "Value": DEVICE})  
 rows.append({"Parameter": "policy", "Value": MLP\_POLICY})  
 rows.append({"Parameter": "checkpoint\_freq\_episodes", "Value": str(CHECKPOINT\_FREQ\_EPISODES)})  
 rows.append({"Parameter": "eval\_freq\_timesteps", "Value": str(EVAL\_FREQ\_TIMESTEPS)})  
 rows.append({"Parameter": "eval\_n\_episodes", "Value": str(EVAL\_N\_EPISODES)})  
  
 print(f"\*\*\* {algo\_name.upper()} Hyperparameters \*\*\*")  
 print(pd.DataFrame(rows).to\_string(index=False))  
 print()

\*\*\* DQN Hyperparameters \*\*\*  
 Parameter Value  
 policy MlpPolicy  
 learning\_rate <function linear\_schedule.<locals>.func at 0x7058383c3560>  
 learning\_starts 50000  
 buffer\_size 750000  
 batch\_size 128  
 gamma 0.99  
 exploration\_fraction 0.12  
 exploration\_final\_eps 0.1  
 target\_update\_interval 250  
 train\_freq 4  
 gradient\_steps 4  
 policy\_kwargs {'net\_arch': [256, 256]}  
 device cpu  
 total\_timesteps 1500000  
 device cpu  
 policy MlpPolicy  
checkpoint\_freq\_episodes 100  
 eval\_freq\_timesteps 25000  
 eval\_n\_episodes 20  
  
\*\*\* PPO Hyperparameters \*\*\*  
 Parameter Value  
 learning\_rate 0.00025  
 n\_steps 2048  
 batch\_size 64  
 n\_epochs 10  
 gamma 0.999  
 gae\_lambda 0.95  
 ent\_coef 0.01  
 clip\_range 0.2  
 total\_timesteps 1500000  
 device cpu  
 policy MlpPolicy  
checkpoint\_freq\_episodes 100  
 eval\_freq\_timesteps 25000  
 eval\_n\_episodes 20

# Recovery: Reconstruct Best-Model Summary from saved eval logs  
# This cell is standalone — it scans models/ folders and rebuilds the table  
# from evaluations.npz files, without needing a prior training run in memory.  
#  
# Results are GROUPED BY SESSION (lab prefix), so seeds trained together  
# in the same notebook run are shown together.  
  
import glob  
  
models\_root = os.path.join(NOTEBOOK\_DIR, "../../../models")  
  
# Collect all completed run data  
all\_runs = [] # list of dicts with session, algo, seed, scores, folder  
  
for algo\_name in ["dqn", "ppo"]:  
 algo\_dir = os.path.join(models\_root, algo\_name)  
 if not os.path.isdir(algo\_dir):  
 continue  
  
 for run\_folder in sorted(glob.glob(os.path.join(algo\_dir, "????-??-??\_??\_??\_??"))):  
 best\_model\_path = os.path.join(run\_folder, "best\_model.zip")  
 eval\_log\_path = os.path.join(run\_folder, "eval\_log", "evaluations.npz")  
  
 if not os.path.isfile(best\_model\_path):  
 continue # incomplete run, skip  
  
 # Extract session (lab prefix) and seed from the final model filename  
 # e.g. lab009\_dqn\_42.zip -> session="lab009", seed="42"  
 session = "unknown"  
 seed\_str = "?"  
 for f in os.listdir(run\_folder):  
 if f.startswith("lab") and f.endswith(".zip") and f != "best\_model.zip":  
 parts = f.replace(".zip", "").split("\_")  
 # parts = ["lab009", "dqn", "42"]  
 session = parts[0] if len(parts) >= 1 else "unknown"  
 seed\_str = parts[-1] if len(parts) >= 3 else "?"  
 break  
  
 timestamp = os.path.basename(run\_folder)  
  
 run\_entry = {  
 "session": session,  
 "algo": algo\_name.upper(),  
 "seed": seed\_str,  
 "timestamp": timestamp,  
 "folder": f"models/{algo\_name}/{timestamp}/",  
 }  
  
 if os.path.isfile(eval\_log\_path):  
 data = np.load(eval\_log\_path, allow\_pickle=True)  
 timesteps = data["timesteps"]  
 results = data["results"]  
  
 best\_score = -np.inf  
 best\_idx = 0  
 for i in range(len(timesteps)):  
 ep\_rewards = results[i]  
 score = np.mean(ep\_rewards) - np.std(ep\_rewards)  
 if score > best\_score:  
 best\_score = score  
 best\_idx = i  
  
 ep = results[best\_idx]  
 run\_entry.update({ # type: ignore  
 "mean\_reward": np.mean(ep),  
 "std\_reward": np.std(ep),  
 "success": np.sum(ep >= 200) / len(ep) \* 100,  
 "score": best\_score,  
 "timestep": int(timesteps[best\_idx]),  
 "has\_eval": True,  
 })  
 else:  
 run\_entry["has\_eval"] = False # type: ignore  
  
 all\_runs.append(run\_entry)  
  
# Group by session  
sessions = sorted(set(r["session"] for r in all\_runs))  
  
if not all\_runs:  
 print("No completed training runs found in models/.")  
else:  
 for session in sessions:  
 session\_runs = [r for r in all\_runs if r["session"] == session]  
  
 print(f"{'='\*70}")  
 print(f"SESSION: {session}")  
 print(f"{'='\*70}")  
  
 rows = []  
 for r in sorted(session\_runs, key=lambda x: (x["algo"], x["seed"])):  
 if r["has\_eval"]:  
 rows.append({  
 "Algorithm": r["algo"],  
 "Seed": r["seed"],  
 "Mean Reward": f"{r['mean\_reward']:.2f}",  
 "Std Reward": f"{r['std\_reward']:.2f}",  
 "Success": f"{r['success']:.0f}%",  
 "Score (mean-std)": f"{r['score']:.2f}",  
 "@ Timestep": f"{r['timestep']:,}",  
 "Run Folder": r["folder"],  
 })  
 else:  
 rows.append({  
 "Algorithm": r["algo"],  
 "Seed": r["seed"],  
 "Mean Reward": "N/A",  
 "Std Reward": "N/A",  
 "Success": "N/A",  
 "Score (mean-std)": "N/A",  
 "@ Timestep": "N/A",  
 "Run Folder": r["folder"] + " (no eval log)",  
 })  
  
 print(pd.DataFrame(rows).to\_string(index=False))  
 print(f"\nRuns in this session: {len(session\_runs)}")  
 print(f"Each 'Run Folder' contains: best\_model.zip, checkpoints/, eval\_log/")  
 print()

======================================================================  
SESSION: lab008  
======================================================================  
Algorithm Seed Mean Reward Std Reward Success Score (mean-std) @ Timestep Run Folder  
 DQN 123 281.04 15.87 100% 265.17 1,225,000 models/dqn/2026-02-20\_19\_35\_00/  
 DQN 42 288.80 12.56 100% 276.24 1,425,000 models/dqn/2026-02-20\_18\_14\_51/  
 DQN 999 288.09 15.54 100% 272.54 1,000,000 models/dqn/2026-02-20\_20\_44\_57/  
 PPO 123 279.66 8.66 100% 271.00 1,375,000 models/ppo/2026-02-20\_22\_53\_14/  
 PPO 42 277.43 12.87 100% 264.56 1,150,000 models/ppo/2026-02-20\_21\_57\_35/  
 PPO 999 289.56 17.39 100% 272.17 1,350,000 models/ppo/2026-02-20\_23\_52\_44/  
  
Runs in this session: 6  
Each 'Run Folder' contains: best\_model.zip, checkpoints/, eval\_log/  
  
======================================================================  
SESSION: lab009  
======================================================================  
Algorithm Seed Mean Reward Std Reward Success Score (mean-std) @ Timestep Run Folder  
 DQN 123 275.61 15.80 100% 259.82 875,000 models/dqn/2026-02-21\_12\_32\_15/  
 DQN 123 275.61 15.80 100% 259.82 875,000 models/dqn/2026-02-21\_15\_59\_14/  
 DQN 123 275.61 15.80 100% 259.82 875,000 models/dqn/2026-02-21\_21\_35\_46/  
 DQN 42 283.33 16.29 100% 267.04 1,450,000 models/dqn/2026-02-21\_05\_29\_23/  
 DQN 42 283.74 17.39 100% 266.35 1,425,000 models/dqn/2026-02-21\_11\_41\_19/  
 DQN 42 283.74 17.39 100% 266.35 1,425,000 models/dqn/2026-02-21\_15\_08\_14/  
 DQN 42 283.74 17.39 100% 266.35 1,425,000 models/dqn/2026-02-21\_20\_43\_20/  
 DQN 999 266.25 22.38 100% 243.87 550,000 models/dqn/2026-02-21\_13\_20\_30/  
 DQN 999 266.25 22.38 100% 243.87 550,000 models/dqn/2026-02-21\_16\_50\_05/  
 DQN 999 266.25 22.38 100% 243.87 550,000 models/dqn/2026-02-21\_22\_27\_19/  
 PPO 123 -968.15 319.34 0% -1287.49 150,000 models/ppo/2026-02-21\_14\_33\_00/  
 PPO 123 -968.15 319.34 0% -1287.49 150,000 models/ppo/2026-02-21\_18\_03\_12/  
 PPO 123 267.61 18.14 100% 249.47 1,300,000 models/ppo/2026-02-21\_23\_39\_19/  
 PPO 42 -868.29 300.16 0% -1168.45 100,000 models/ppo/2026-02-21\_14\_09\_36/  
 PPO 42 -868.29 300.16 0% -1168.45 100,000 models/ppo/2026-02-21\_17\_40\_16/  
 PPO 42 289.50 21.48 100% 268.02 1,500,000 models/ppo/2026-02-21\_23\_17\_45/  
 PPO 999 -794.03 451.90 0% -1245.93 250,000 models/ppo/2026-02-21\_18\_25\_37/  
 PPO 999 285.68 19.69 100% 266.00 1,175,000 models/ppo/2026-02-22\_00\_03\_53/  
  
Runs in this session: 18  
Each 'Run Folder' contains: best\_model.zip, checkpoints/, eval\_log/  
  
======================================================================  
SESSION: unknown  
======================================================================  
Algorithm Seed Mean Reward Std Reward Success Score (mean-std) @ Timestep Run Folder  
 DQN ? 256.82 20.28 100% 236.55 125,000 models/dqn/2026-02-21\_07\_02\_05/  
 PPO ? -794.03 451.90 0% -1245.93 250,000 models/ppo/2026-02-21\_14\_55\_39/  
  
Runs in this session: 2  
Each 'Run Folder' contains: best\_model.zip, checkpoints/, eval\_log/

# DQN & PPO Multi-Seed Report (Best Model Selection)

This notebook loads the **best model** from each training run (selected by the combined metric mean\_reward - std\_reward during training) and runs evaluation and visualization.

No training is required — run lab009\_v1.ipynb first to generate the model files.

Models are loaded from timestamped run folders: models/{algo}/{timestamp}/best\_model.zip

import os, sys  
  
import numpy as np  
import pandas as pd  
import torch  
import matplotlib.pyplot as plt  
from scipy import stats  
  
import gymnasium as gym  
from stable\_baselines3 import DQN, PPO  
from stable\_baselines3.common.evaluation import evaluate\_policy  
from stable\_baselines3.common.vec\_env import DummyVecEnv  
from stable\_baselines3.common.monitor import Monitor  
  
import imageio  
from IPython.display import Image, display

# Configuration  
  
import glob  
  
SEED\_LIST = [42, 123, 999]  
  
ALGORITHM\_MAP = {  
 "dqn": DQN,  
 "ppo": PPO,  
}  
  
NOTEBOOK\_DIR = os.path.dirname(os.path.abspath("\_\_file\_\_"))  
GYMNASIUM\_MODEL = "LunarLander-v3"  
  
WIND\_ENABLED = False  
  
EVALUATION\_EPISODES = 20  
  
TRAJECTORY\_EPISODES = 3 # Episodes to visualize per algorithm for trajectory plots  
  
DEVICE = "cpu"  
  
# Session prefix — must match the final model filenames from training  
# (e.g. lab009\_dqn\_42.zip -> SESSION\_PREFIX = "lab009")  
SESSION\_PREFIX = "lab009"  
  
# LunarLander-v3 action labels  
ACTION\_LABELS = ["Do Nothing", "Fire Left", "Fire Main", "Fire Right"]  
  
  
def discover\_best\_models(session\_prefix):  
 """  
 Scan models/{algo}/{timestamp}/ folders and return a dict:  
 {algo: {seed: path\_to\_best\_model}}  
 Only considers runs whose final model filename starts with session\_prefix.  
 """  
 models\_root = os.path.join(NOTEBOOK\_DIR, "../../../models")  
 best\_models = {}  
  
 for algo\_name in ALGORITHM\_MAP:  
 best\_models[algo\_name] = {}  
 algo\_dir = os.path.join(models\_root, algo\_name)  
 if not os.path.isdir(algo\_dir):  
 continue  
  
 for run\_folder in sorted(glob.glob(os.path.join(algo\_dir, "????-??-??\_??\_??\_??"))):  
 best\_model\_path = os.path.join(run\_folder, "best\_model.zip")  
 if not os.path.isfile(best\_model\_path):  
 continue  
  
 # Find the final model file to extract session and seed  
 for f in os.listdir(run\_folder):  
 if f.startswith(session\_prefix) and f.endswith(".zip") and f != "best\_model.zip":  
 seed\_str = f.replace(".zip", "").split("\_")[-1]  
 if seed\_str.isdigit():  
 seed\_int = int(seed\_str)  
 if seed\_int in SEED\_LIST:  
 best\_models[algo\_name][seed\_int] = best\_model\_path  
 break  
  
 return best\_models  
  
  
# Discover best models for this session  
best\_model\_paths = discover\_best\_models(SESSION\_PREFIX)  
  
print(f"Session: {SESSION\_PREFIX}")  
print(f"Algorithms: {list(ALGORITHM\_MAP.keys())}")  
print(f"Seeds: {SEED\_LIST}")  
print(f"Wind enabled: {WIND\_ENABLED}")  
print(f"Evaluation episodes per seed: {EVALUATION\_EPISODES}")  
print(f"Device: {DEVICE}")  
print()  
print("Discovered best models:")  
for algo\_name in ALGORITHM\_MAP:  
 for seed in SEED\_LIST:  
 path = best\_model\_paths.get(algo\_name, {}).get(seed)  
 status = path if path else "NOT FOUND"  
 print(f" {algo\_name.upper()} seed {seed}: {status}")

Session: lab009  
Algorithms: ['dqn', 'ppo']  
Seeds: [42, 123, 999]  
Wind enabled: False  
Evaluation episodes per seed: 20  
Device: cpu  
  
Discovered best models:  
 DQN seed 42: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_20\_43\_20/best\_model.zip  
 DQN seed 123: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_21\_35\_46/best\_model.zip  
 DQN seed 999: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/dqn/2026-02-21\_22\_27\_19/best\_model.zip  
 PPO seed 42: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_17\_45/best\_model.zip  
 PPO seed 123: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-21\_23\_39\_19/best\_model.zip  
 PPO seed 999: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/../../../models/ppo/2026-02-22\_00\_03\_53/best\_model.zip

# Load all best models and evaluate  
  
evaluation\_results = {} # {algo: {seed: np.array}}  
  
for algo\_name, algo\_class in ALGORITHM\_MAP.items():  
 evaluation\_results[algo\_name] = {}  
  
 for seed in SEED\_LIST:  
 load\_path = best\_model\_paths.get(algo\_name, {}).get(seed)  
 if load\_path is None:  
 print(f"SKIPPING {algo\_name.upper()} seed {seed} — best model not found")  
 continue  
  
 print(f"Loading and evaluating {algo\_name.upper()} seed {seed} (best model)...")  
  
 def make\_env(s=seed):  
 env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 env.reset(seed=s)  
 return env  
  
 model = algo\_class.load(load\_path, env=DummyVecEnv([make\_env]), device=DEVICE)  
  
 eval\_env = Monitor(gym.make(GYMNASIUM\_MODEL, enable\_wind=WIND\_ENABLED))  
 eval\_env.reset(seed=seed)  
  
 rewards, \_ = evaluate\_policy(  
 model,  
 eval\_env,  
 n\_eval\_episodes=EVALUATION\_EPISODES,  
 deterministic=True,  
 return\_episode\_rewards=True  
 )  
  
 evaluation\_results[algo\_name][seed] = np.array(rewards)  
 eval\_env.close()  
  
 print(f"{algo\_name.upper()}: evaluation complete.\n")  
  
print(f"All evaluations complete.")

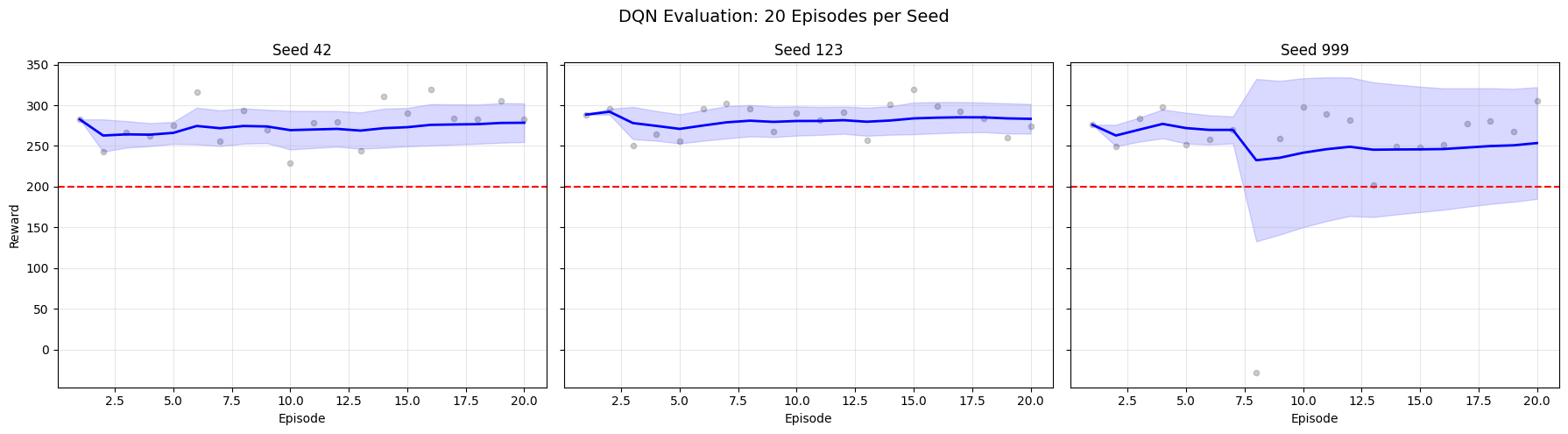
Loading and evaluating DQN seed 42 (best model)...  
Loading and evaluating DQN seed 123 (best model)...  
Loading and evaluating DQN seed 999 (best model)...  
DQN: evaluation complete.  
  
Loading and evaluating PPO seed 42 (best model)...  
Loading and evaluating PPO seed 123 (best model)...  
Loading and evaluating PPO seed 999 (best model)...  
PPO: evaluation complete.  
  
All evaluations complete.

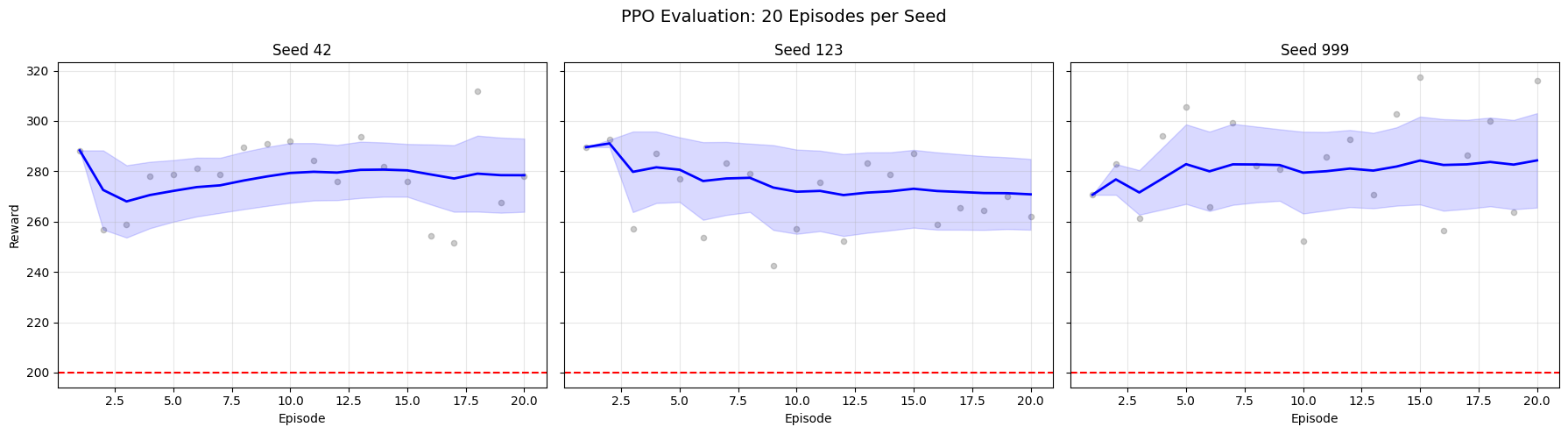
## Per-Algorithm Results

# Per-Algorithm: Evaluation Summary Tables  
  
for algo\_name in ALGORITHM\_MAP:  
 rows = []  
 for seed in SEED\_LIST:  
 r = evaluation\_results[algo\_name][seed]  
 rows.append({  
 "Seed": seed,  
 "Mean Reward": f"{np.mean(r):.2f}",  
 "Std Dev": f"{np.std(r):.2f}",  
 "Min Reward": f"{np.min(r):.2f}",  
 "Max Reward": f"{np.max(r):.2f}",  
 "Success Rate": f"{(r >= 200).sum() / len(r) \* 100:.1f}%"  
 })  
  
 all\_r = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
 rows.append({  
 "Seed": "Overall",  
 "Mean Reward": f"{np.mean(all\_r):.2f}",  
 "Std Dev": f"{np.std(all\_r):.2f}",  
 "Min Reward": f"{np.min(all\_r):.2f}",  
 "Max Reward": f"{np.max(all\_r):.2f}",  
 "Success Rate": f"{(all\_r >= 200).sum() / len(all\_r) \* 100:.1f}%"  
 })  
  
 print(f"\*\*\* {algo\_name.upper()} MULTI-SEED EVALUATION SUMMARY \*\*\*")  
 print(f"Episodes per seed: {EVALUATION\_EPISODES} | Total: {len(all\_r)}")  
 print(pd.DataFrame(rows).to\_string(index=False))  
 print()

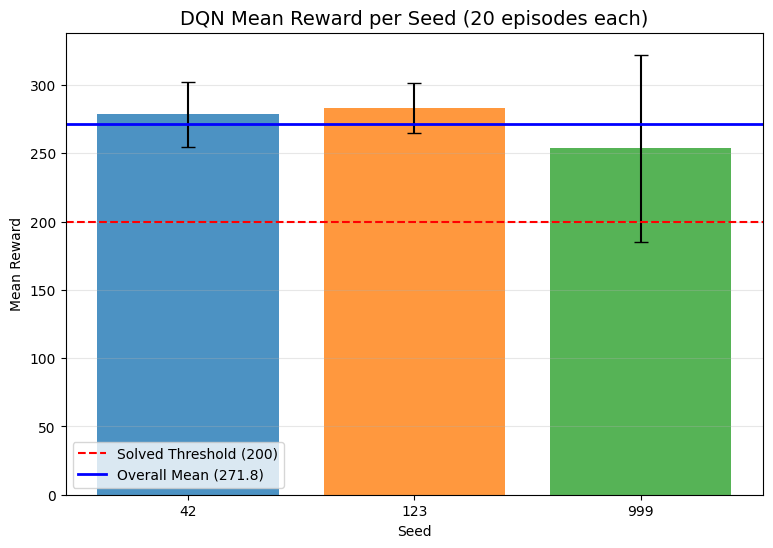
\*\*\* DQN MULTI-SEED EVALUATION SUMMARY \*\*\*  
Episodes per seed: 20 | Total: 60  
 Seed Mean Reward Std Dev Min Reward Max Reward Success Rate  
 42 278.51 23.70 228.67 318.97 100.0%  
 123 283.34 18.19 250.33 319.93 100.0%  
 999 253.57 68.63 -28.26 305.91 95.0%  
Overall 271.80 45.14 -28.26 319.93 98.3%  
  
\*\*\* PPO MULTI-SEED EVALUATION SUMMARY \*\*\*  
Episodes per seed: 20 | Total: 60  
 Seed Mean Reward Std Dev Min Reward Max Reward Success Rate  
 42 278.43 14.56 251.45 311.70 100.0%  
 123 270.83 14.06 242.53 292.55 100.0%  
 999 284.32 18.77 252.25 317.36 100.0%  
Overall 277.86 16.87 242.53 317.36 100.0%

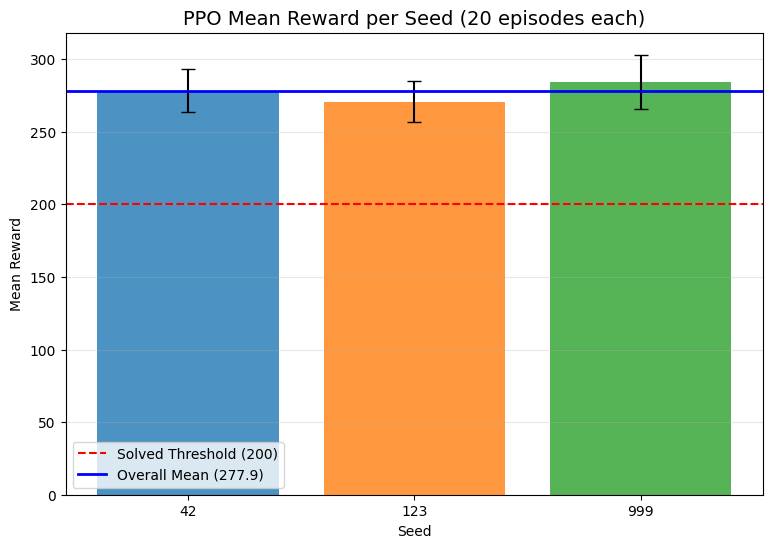
# Per-Algorithm, Per-Seed: Evaluation Convergence Plots  
  
for algo\_name in ALGORITHM\_MAP:  
 fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
 if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
 for ax, seed in zip(axes, SEED\_LIST):  
 rewards = evaluation\_results[algo\_name][seed]  
 episodes = np.arange(1, len(rewards) + 1)  
 running\_mean = np.cumsum(rewards) / episodes  
 running\_std = np.array([np.std(rewards[:i]) for i in episodes])  
  
 ax.scatter(episodes, rewards, color='gray', alpha=0.4, s=20, label='Episode Reward')  
 ax.plot(episodes, running\_mean, color='blue', linewidth=2, label='Running Mean')  
 ax.fill\_between(episodes, running\_mean - running\_std, running\_mean + running\_std,  
 color='blue', alpha=0.15)  
 ax.axhline(y=200, color='red', linestyle='--')  
 ax.set\_title(f"Seed {seed}")  
 ax.set\_xlabel("Episode")  
 ax.grid(True, alpha=0.3)  
  
 axes[0].set\_ylabel("Reward")  
 fig.suptitle(f"{algo\_name.upper()} Evaluation: {EVALUATION\_EPISODES} Episodes per Seed", fontsize=14)  
 plt.tight\_layout()  
 plt.show()



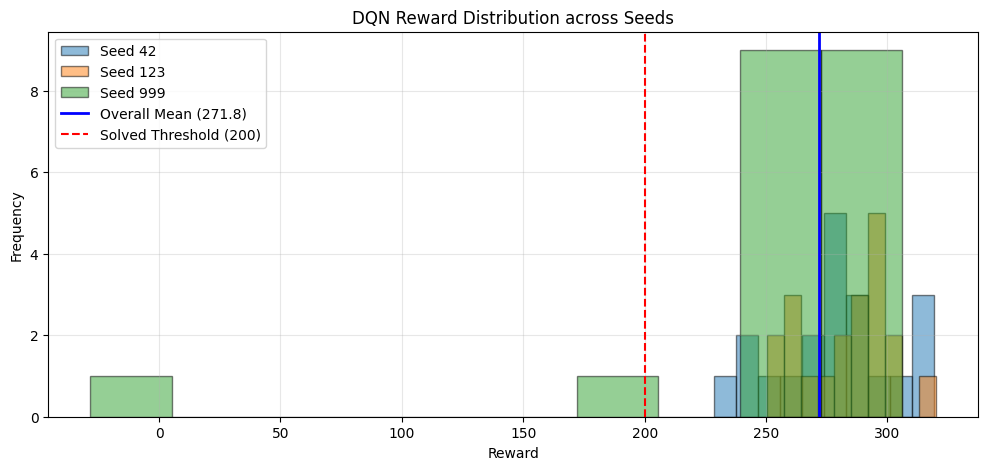


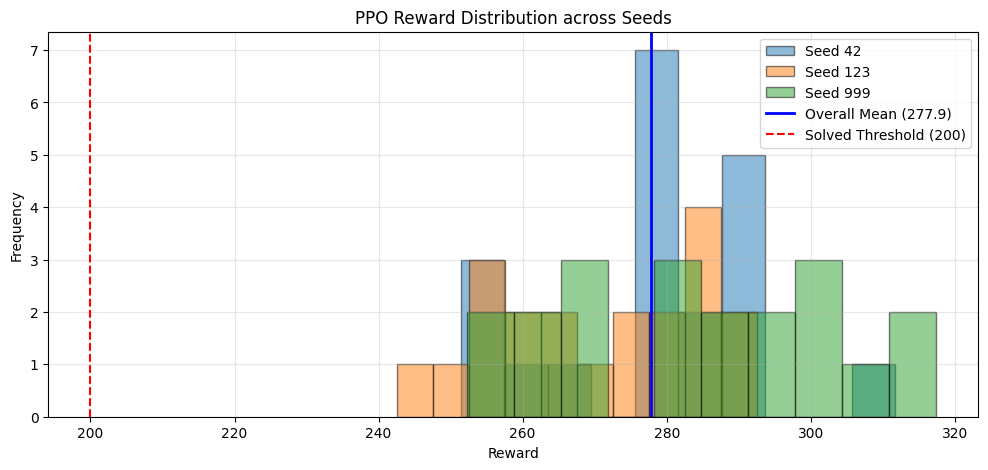
# Per-Algorithm: Evaluation Bar Chart (mean reward per seed with error bars)  
  
seed\_colors = list(plt.colormaps["tab10"](range(10))) # type: ignore[arg-type]  
  
for algo\_name in ALGORITHM\_MAP:  
 all\_r = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
 means = [np.mean(evaluation\_results[algo\_name][s]) for s in SEED\_LIST]  
 stds = [np.std(evaluation\_results[algo\_name][s]) for s in SEED\_LIST]  
 labels = [str(s) for s in SEED\_LIST]  
  
 plt.figure(figsize=(max(8, 3 \* len(SEED\_LIST)), 6))  
 plt.bar(labels, means, yerr=stds, capsize=5, color=seed\_colors[:len(SEED\_LIST)], alpha=0.8)  
 plt.axhline(y=200, color='red', linestyle='--', label='Solved Threshold (200)')  
 plt.axhline(y=float(np.mean(all\_r)), color='blue', linestyle='-', linewidth=2,  
 label=f'Overall Mean ({np.mean(all\_r):.1f})')  
  
 plt.title(f"{algo\_name.upper()} Mean Reward per Seed ({EVALUATION\_EPISODES} episodes each)", fontsize=14)  
 plt.xlabel("Seed")  
 plt.ylabel("Mean Reward")  
 plt.legend()  
 plt.grid(True, alpha=0.3, axis='y')  
 plt.show()





# Per-Algorithm: Reward Distribution Histograms (overlaid per seed)  
  
for algo\_name in ALGORITHM\_MAP:  
 all\_r = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
  
 plt.figure(figsize=(12, 5))  
 for i, seed in enumerate(SEED\_LIST):  
 plt.hist(evaluation\_results[algo\_name][seed], bins=10, alpha=0.5,  
 color=seed\_colors[i], edgecolor='black', label=f"Seed {seed}")  
  
 plt.axvline(x=float(np.mean(all\_r)), color='blue', linestyle='-', linewidth=2,  
 label=f'Overall Mean ({np.mean(all\_r):.1f})')  
 plt.axvline(x=200, color='red', linestyle='--', label='Solved Threshold (200)')  
 plt.title(f'{algo\_name.upper()} Reward Distribution across Seeds')  
 plt.xlabel('Reward')  
 plt.ylabel('Frequency')  
 plt.legend()  
 plt.grid(True, alpha=0.3)  
 plt.show()



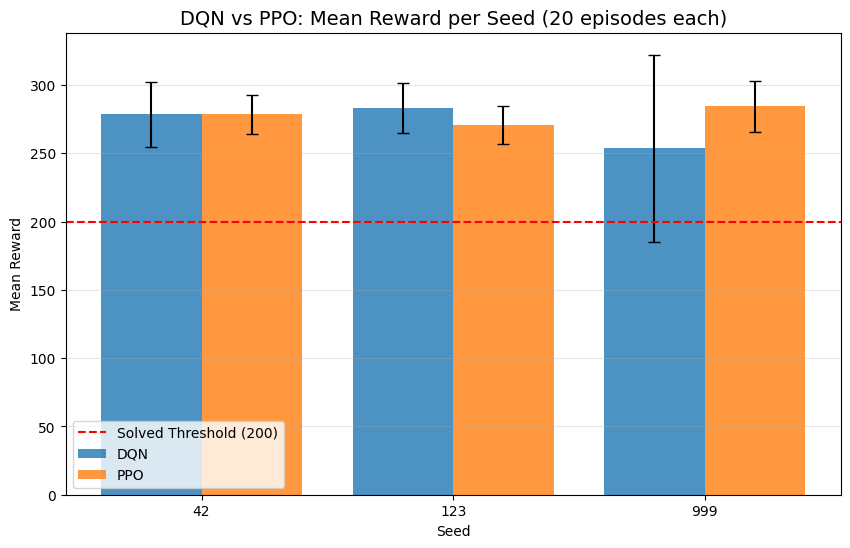


## Cross-Algorithm Comparison

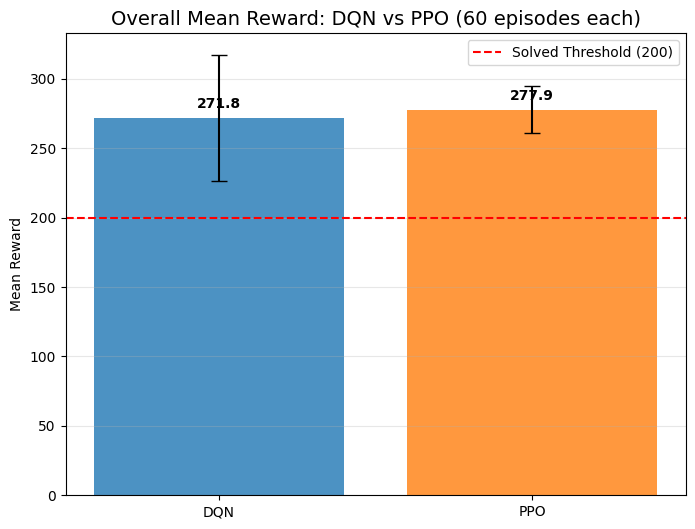
# Cross-Algorithm: Combined Summary Table  
  
rows = []  
for algo\_name in ALGORITHM\_MAP:  
 all\_r = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
 rows.append({  
 "Algorithm": algo\_name.upper(),  
 "Mean Reward": f"{np.mean(all\_r):.2f}",  
 "Std Dev": f"{np.std(all\_r):.2f}",  
 "Min Reward": f"{np.min(all\_r):.2f}",  
 "Max Reward": f"{np.max(all\_r):.2f}",  
 "Success Rate": f"{(all\_r >= 200).sum() / len(all\_r) \* 100:.1f}%"  
 })  
  
print(f"\*\*\* CROSS-ALGORITHM EVALUATION SUMMARY \*\*\*")  
print(f"Seeds: {SEED\_LIST} | Episodes per seed: {EVALUATION\_EPISODES}")  
print(f"Total episodes per algorithm: {EVALUATION\_EPISODES \* len(SEED\_LIST)}")  
print()  
print(pd.DataFrame(rows).to\_string(index=False))

\*\*\* CROSS-ALGORITHM EVALUATION SUMMARY \*\*\*  
Seeds: [42, 123, 999] | Episodes per seed: 20  
Total episodes per algorithm: 60  
  
Algorithm Mean Reward Std Dev Min Reward Max Reward Success Rate  
 DQN 271.80 45.14 -28.26 319.93 98.3%  
 PPO 277.86 16.87 242.53 317.36 100.0%

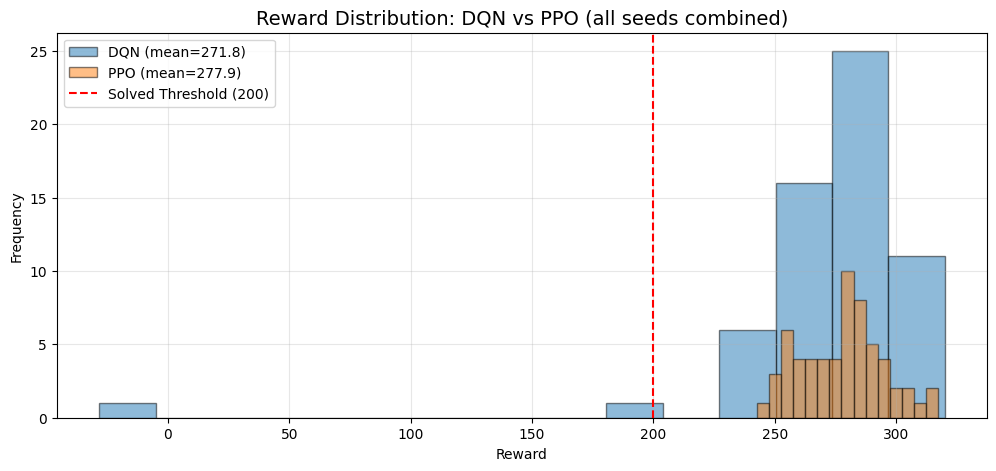
# Cross-Algorithm: Grouped Bar Chart (DQN vs PPO per seed)  
  
algo\_names = list(ALGORITHM\_MAP.keys())  
n\_algos = len(algo\_names)  
n\_seeds = len(SEED\_LIST)  
bar\_width = 0.8 / n\_algos  
x = np.arange(n\_seeds)  
  
plt.figure(figsize=(max(10, 3 \* n\_seeds), 6))  
for i, algo\_name in enumerate(algo\_names):  
 means = [np.mean(evaluation\_results[algo\_name][s]) for s in SEED\_LIST]  
 stds = [np.std(evaluation\_results[algo\_name][s]) for s in SEED\_LIST]  
 offset = (i - (n\_algos - 1) / 2) \* bar\_width  
 plt.bar(x + offset, means, bar\_width, yerr=stds, capsize=4,  
 label=algo\_name.upper(), alpha=0.8)  
  
plt.axhline(y=200, color='red', linestyle='--', label='Solved Threshold (200)')  
plt.xticks(x, [str(s) for s in SEED\_LIST])  
plt.title(f"DQN vs PPO: Mean Reward per Seed ({EVALUATION\_EPISODES} episodes each)", fontsize=14)  
plt.xlabel("Seed")  
plt.ylabel("Mean Reward")  
plt.legend()  
plt.grid(True, alpha=0.3, axis='y')  
plt.show()



# Cross-Algorithm: Overall Mean Reward Bar Chart  
  
algo\_colors = {"dqn": "tab:blue", "ppo": "tab:orange"}  
  
overall\_means = []  
overall\_stds = []  
for algo\_name in algo\_names:  
 all\_r = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
 overall\_means.append(np.mean(all\_r))  
 overall\_stds.append(np.std(all\_r))  
  
plt.figure(figsize=(8, 6))  
bars = plt.bar([a.upper() for a in algo\_names], overall\_means, yerr=overall\_stds,  
 capsize=6, color=[algo\_colors[a] for a in algo\_names], alpha=0.8)  
plt.axhline(y=200, color='red', linestyle='--', label='Solved Threshold (200)')  
  
for bar, mean in zip(bars, overall\_means):  
 plt.text(bar.get\_x() + bar.get\_width() / 2, bar.get\_height() + 5,  
 f'{mean:.1f}', ha='center', va='bottom', fontweight='bold')  
  
plt.title(f"Overall Mean Reward: DQN vs PPO ({EVALUATION\_EPISODES \* len(SEED\_LIST)} episodes each)", fontsize=14)  
plt.ylabel("Mean Reward")  
plt.legend()  
plt.grid(True, alpha=0.3, axis='y')  
plt.show()

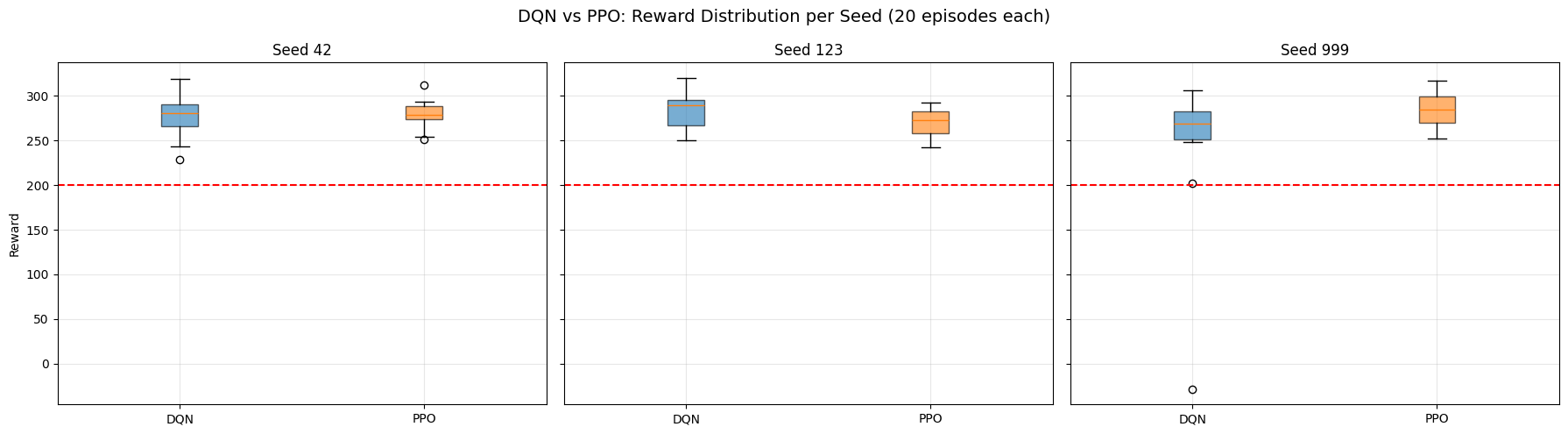


# Cross-Algorithm: Reward Distribution Comparison (overlaid histograms)  
  
plt.figure(figsize=(12, 5))  
for algo\_name in algo\_names:  
 all\_r = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
 plt.hist(all\_r, bins=15, alpha=0.5, color=algo\_colors[algo\_name],  
 edgecolor='black', label=f"{algo\_name.upper()} (mean={np.mean(all\_r):.1f})")  
  
plt.axvline(x=200, color='red', linestyle='--', label='Solved Threshold (200)')  
plt.title('Reward Distribution: DQN vs PPO (all seeds combined)', fontsize=14)  
plt.xlabel('Reward')  
plt.ylabel('Frequency')  
plt.legend()  
plt.grid(True, alpha=0.3)  
plt.show()



# Cross-Algorithm: Box Plot Comparison per Seed  
  
fig, axes = plt.subplots(1, len(SEED\_LIST), figsize=(6 \* len(SEED\_LIST), 5), sharey=True)  
if len(SEED\_LIST) == 1:  
 axes = [axes]  
  
for ax, seed in zip(axes, SEED\_LIST):  
 data = [evaluation\_results[algo\_name][seed] for algo\_name in algo\_names]  
 bp = ax.boxplot(data, labels=[a.upper() for a in algo\_names], patch\_artist=True)  
 for patch, algo\_name in zip(bp['boxes'], algo\_names):  
 patch.set\_facecolor(algo\_colors[algo\_name])  
 patch.set\_alpha(0.6)  
 ax.axhline(y=200, color='red', linestyle='--')  
 ax.set\_title(f"Seed {seed}")  
 ax.grid(True, alpha=0.3)  
  
axes[0].set\_ylabel("Reward")  
fig.suptitle(f"DQN vs PPO: Reward Distribution per Seed ({EVALUATION\_EPISODES} episodes each)", fontsize=14)  
plt.tight\_layout()  
plt.show()

/tmp/ipykernel\_86514/3208474440.py:9: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick\_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.  
 bp = ax.boxplot(data, labels=[a.upper() for a in algo\_names], patch\_artist=True)  
/tmp/ipykernel\_86514/3208474440.py:9: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick\_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.  
 bp = ax.boxplot(data, labels=[a.upper() for a in algo\_names], patch\_artist=True)  
/tmp/ipykernel\_86514/3208474440.py:9: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick\_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.  
 bp = ax.boxplot(data, labels=[a.upper() for a in algo\_names], patch\_artist=True)



## Statistical Significance

# Statistical Significance: Mann-Whitney U Tests  
  
algo\_names = list(ALGORITHM\_MAP.keys())  
  
# Gather all rewards per algorithm  
algo\_all\_rewards = {}  
for algo\_name in algo\_names:  
 algo\_all\_rewards[algo\_name] = np.concatenate([evaluation\_results[algo\_name][s] for s in SEED\_LIST])  
  
# --- Reward comparison (Mann-Whitney U) ---  
mwu\_result = stats.mannwhitneyu(  
 algo\_all\_rewards[algo\_names[0]],  
 algo\_all\_rewards[algo\_names[1]],  
 alternative='two-sided'  
)  
stat\_reward = float(mwu\_result.statistic)  
p\_reward = float(mwu\_result.pvalue)  
  
# --- Success rate comparison (Chi-squared) ---  
successes = []  
totals = []  
for algo\_name in algo\_names:  
 r = algo\_all\_rewards[algo\_name]  
 successes.append(int((r >= 200).sum()))  
 totals.append(len(r))  
  
failures = [t - s for t, s in zip(totals, successes)]  
contingency = np.array([successes, failures])  
  
# Chi-squared requires all expected frequencies > 0; skip if any row/col is all-zero  
if np.all(contingency.sum(axis=1) > 0) and np.all(contingency.sum(axis=0) > 0):  
 chi2\_result = stats.chi2\_contingency(contingency)  
 chi2 = float(chi2\_result[0]) # type: ignore[arg-type] # statistic  
 p\_success = float(chi2\_result[1]) # type: ignore[arg-type] # pvalue  
 chi2\_valid = True  
else:  
 chi2, p\_success = 0.0, 1.0  
 chi2\_valid = False  
  
# --- Results table ---  
chi2\_note = "" if chi2\_valid else " (skipped: zero row/col)"  
rows = [  
 {  
 "Metric": "Mean Reward",  
 f"{algo\_names[0].upper()} Value": f"{np.mean(algo\_all\_rewards[algo\_names[0]]):.2f}",  
 f"{algo\_names[1].upper()} Value": f"{np.mean(algo\_all\_rewards[algo\_names[1]]):.2f}",  
 "Test": "Mann-Whitney U",  
 "Statistic": f"{stat\_reward:.1f}",  
 "p-value": f"{p\_reward:.4f}",  
 "Significant (p<0.05)": "Yes" if p\_reward < 0.05 else "No"  
 },  
 {  
 "Metric": "Success Rate (>=200)",  
 f"{algo\_names[0].upper()} Value": f"{successes[0]/totals[0]\*100:.1f}%",  
 f"{algo\_names[1].upper()} Value": f"{successes[1]/totals[1]\*100:.1f}%",  
 "Test": f"Chi-squared{chi2\_note}",  
 "Statistic": f"{chi2:.2f}",  
 "p-value": f"{p\_success:.4f}",  
 "Significant (p<0.05)": "Yes" if (chi2\_valid and p\_success < 0.05) else "No"  
 },  
]  
  
print("\*\*\* STATISTICAL SIGNIFICANCE TESTS \*\*\*")  
print(f"Sample size per algorithm: {totals[0]} episodes ({EVALUATION\_EPISODES} episodes x {len(SEED\_LIST)} seeds)")  
print()  
print(pd.DataFrame(rows).to\_string(index=False))  
print()  
if not chi2\_valid:  
 print("Note: Chi-squared test skipped because one or both algorithms had 0% or 100% success rate.")  
 print(" This is expected during smoke tests with few episodes. Full run will have enough data.")  
 print()  
if p\_reward < 0.05:  
 print(f"The reward difference between {algo\_names[0].upper()} and {algo\_names[1].upper()} is statistically significant (p={p\_reward:.4f}).")  
else:  
 print(f"No statistically significant reward difference between {algo\_names[0].upper()} and {algo\_names[1].upper()} (p={p\_reward:.4f}).")

\*\*\* STATISTICAL SIGNIFICANCE TESTS \*\*\*  
Sample size per algorithm: 60 episodes (20 episodes x 3 seeds)  
  
 Metric DQN Value PPO Value Test Statistic p-value Significant (p<0.05)  
 Mean Reward 271.80 277.86 Mann-Whitney U 1780.0 0.9185 No  
Success Rate (>=200) 98.3% 100.0% Chi-squared 0.00 1.0000 No  
  
No statistically significant reward difference between DQN and PPO (p=0.9185).

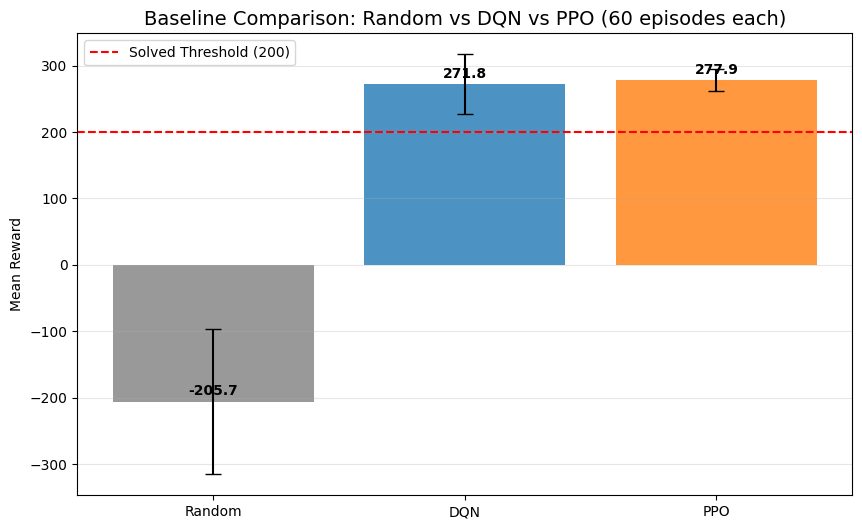
## Baseline Comparison

# Random Agent Baseline Evaluation  
  
random\_results = {}  
  
for seed in SEED\_LIST:  
 print(f"Running random agent with seed {seed}...")  
 env = gym.make(GYMNASIUM\_MODEL, enable\_wind=WIND\_ENABLED)  
 env.action\_space.seed(seed)  
 episode\_rewards = []  
  
 for ep in range(EVALUATION\_EPISODES):  
 obs, info = env.reset(seed=seed + ep)  
 total\_reward = 0.0  
 done = False  
  
 while not done:  
 action = env.action\_space.sample()  
 obs, reward, terminated, truncated, info = env.step(action)  
 total\_reward += float(reward)  
 done = terminated or truncated  
  
 episode\_rewards.append(total\_reward)  
  
 random\_results[seed] = np.array(episode\_rewards)  
 env.close()  
  
print("Random baseline evaluation complete.")

Running random agent with seed 42...  
Running random agent with seed 123...  
Running random agent with seed 999...  
Random baseline evaluation complete.

# Baseline Comparison: Table + Chart  
  
algo\_names = list(ALGORITHM\_MAP.keys())  
algo\_colors = {"dqn": "tab:blue", "ppo": "tab:orange"}  
algo\_all\_rewards = {a: np.concatenate([evaluation\_results[a][s] for s in SEED\_LIST]) for a in algo\_names}  
  
all\_random = np.concatenate([random\_results[s] for s in SEED\_LIST])  
  
rows = [  
 {  
 "Agent": "Random",  
 "Mean Reward": f"{np.mean(all\_random):.2f}",  
 "Std Dev": f"{np.std(all\_random):.2f}",  
 "Min": f"{np.min(all\_random):.2f}",  
 "Max": f"{np.max(all\_random):.2f}",  
 "Success Rate": f"{(all\_random >= 200).sum() / len(all\_random) \* 100:.1f}%"  
 }  
]  
for algo\_name in algo\_names:  
 all\_r = algo\_all\_rewards[algo\_name]  
 rows.append({  
 "Agent": algo\_name.upper(),  
 "Mean Reward": f"{np.mean(all\_r):.2f}",  
 "Std Dev": f"{np.std(all\_r):.2f}",  
 "Min": f"{np.min(all\_r):.2f}",  
 "Max": f"{np.max(all\_r):.2f}",  
 "Success Rate": f"{(all\_r >= 200).sum() / len(all\_r) \* 100:.1f}%"  
 })  
rows.append({  
 "Agent": "Human (ref)",  
 "Mean Reward": "~200-300",  
 "Std Dev": "-",  
 "Min": "-",  
 "Max": "-",  
 "Success Rate": "~100%"  
})  
  
print("\*\*\* BASELINE COMPARISON \*\*\*")  
print(pd.DataFrame(rows).to\_string(index=False))  
print()  
  
# Bar chart  
agent\_labels = ["Random"] + [a.upper() for a in algo\_names]  
agent\_means = [np.mean(all\_random)] + [np.mean(algo\_all\_rewards[a]) for a in algo\_names]  
agent\_stds = [np.std(all\_random)] + [np.std(algo\_all\_rewards[a]) for a in algo\_names]  
bar\_colors = ["gray"] + [algo\_colors[a] for a in algo\_names]  
  
plt.figure(figsize=(10, 6))  
bars = plt.bar(agent\_labels, agent\_means, yerr=agent\_stds, capsize=6,  
 color=bar\_colors, alpha=0.8)  
plt.axhline(y=200, color='red', linestyle='--', label='Solved Threshold (200)')  
  
for bar, mean in zip(bars, agent\_means):  
 plt.text(bar.get\_x() + bar.get\_width() / 2, bar.get\_height() + 5,  
 f'{mean:.1f}', ha='center', va='bottom', fontweight='bold')  
  
plt.title(f"Baseline Comparison: Random vs DQN vs PPO ({EVALUATION\_EPISODES \* len(SEED\_LIST)} episodes each)", fontsize=14)  
plt.ylabel("Mean Reward")  
plt.legend()  
plt.grid(True, alpha=0.3, axis='y')  
plt.show()

\*\*\* BASELINE COMPARISON \*\*\*  
 Agent Mean Reward Std Dev Min Max Success Rate  
 Random -205.65 109.48 -420.28 15.45 0.0%  
 DQN 271.80 45.14 -28.26 319.93 98.3%  
 PPO 277.86 16.87 242.53 317.36 100.0%  
Human (ref) ~200-300 - - - ~100%

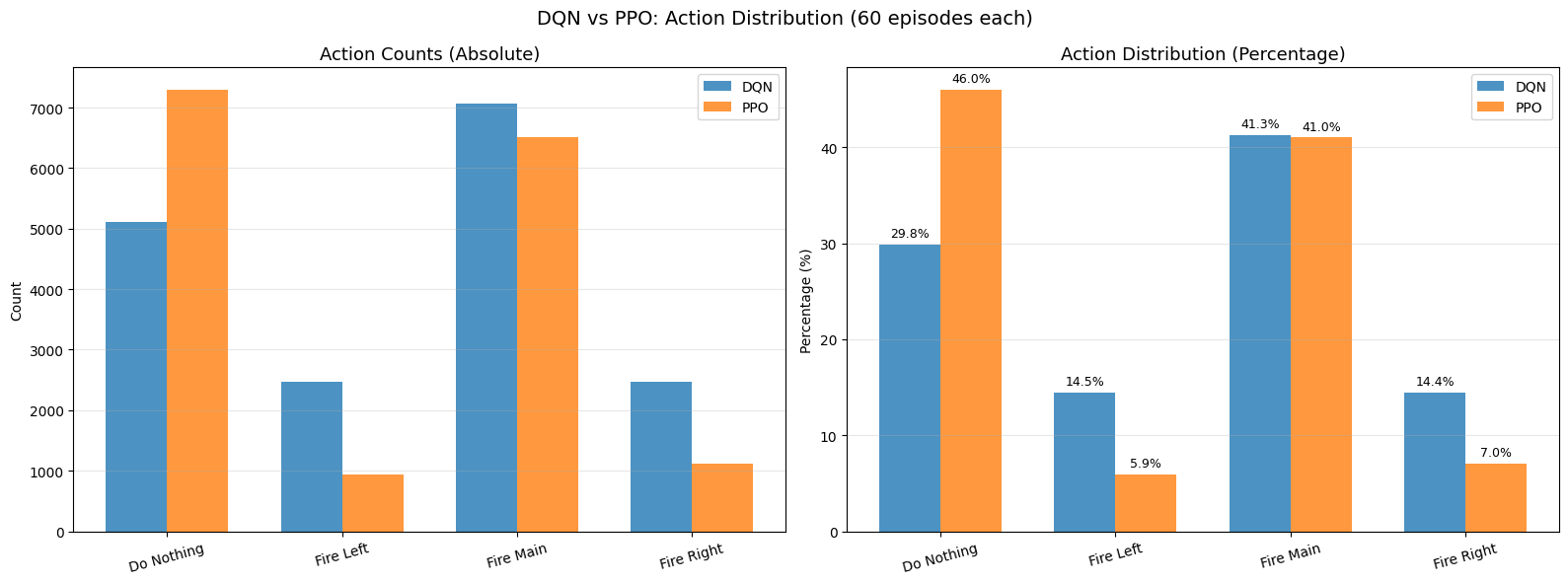


## Agent Behavior Analysis

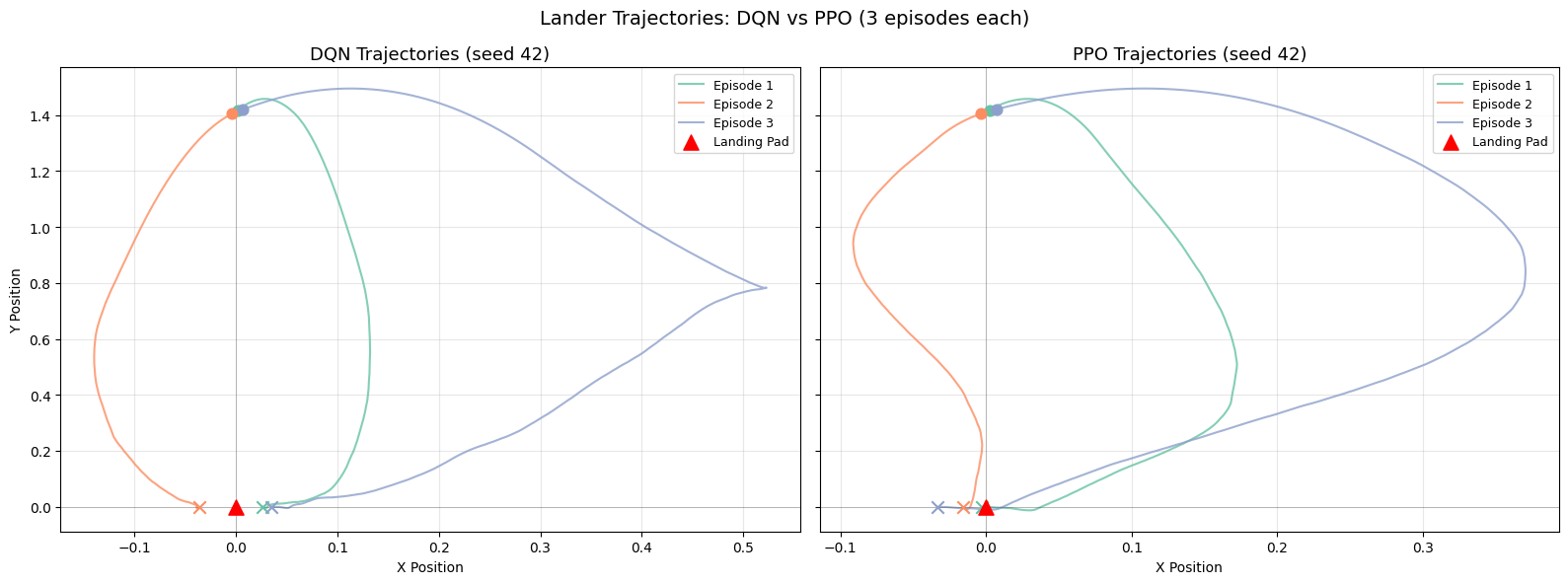
# Collect per-step data: actions and trajectories  
  
action\_counts = {} # {algo: np.array of shape (4,)} total action counts  
trajectory\_data = {} # {algo: list of (x\_positions, y\_positions)} one per TRAJECTORY\_EPISODES  
  
for algo\_name, algo\_class in ALGORITHM\_MAP.items():  
 action\_counts[algo\_name] = np.zeros(len(ACTION\_LABELS), dtype=int)  
 trajectory\_data[algo\_name] = []  
  
 for seed in SEED\_LIST:  
 load\_path = best\_model\_paths.get(algo\_name, {}).get(seed)  
 if load\_path is None:  
 continue  
  
 def make\_env(s=seed):  
 env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 env.reset(seed=s)  
 return env  
  
 model = algo\_class.load(load\_path, env=DummyVecEnv([make\_env]), device=DEVICE)  
 env = gym.make(GYMNASIUM\_MODEL, enable\_wind=WIND\_ENABLED)  
  
 for ep in range(EVALUATION\_EPISODES):  
 obs, info = env.reset(seed=seed + ep)  
 done = False  
 x\_pos, y\_pos = [obs[0]], [obs[1]]  
  
 while not done:  
 action, \_ = model.predict(obs, deterministic=True)  
 action\_int = int(action)  
 action\_counts[algo\_name][action\_int] += 1  
  
 obs, reward, terminated, truncated, info = env.step(action)  
 done = terminated or truncated  
 x\_pos.append(obs[0])  
 y\_pos.append(obs[1])  
  
 # Keep trajectory for the first TRAJECTORY\_EPISODES episodes of the first seed  
 if seed == SEED\_LIST[0] and ep < TRAJECTORY\_EPISODES:  
 trajectory\_data[algo\_name].append((np.array(x\_pos), np.array(y\_pos)))  
  
 env.close()  
  
 total\_actions = action\_counts[algo\_name].sum()  
 print(f"{algo\_name.upper()}: {total\_actions:,} total actions collected across {EVALUATION\_EPISODES \* len(SEED\_LIST)} episodes")  
  
print("\nBehavior data collection complete.")

DQN: 17,103 total actions collected across 60 episodes  
PPO: 15,857 total actions collected across 60 episodes  
  
Behavior data collection complete.

# Action Distribution: DQN vs PPO  
  
n\_actions = len(ACTION\_LABELS)  
x = np.arange(n\_actions)  
bar\_width = 0.35  
  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))  
  
# Absolute counts  
for i, algo\_name in enumerate(algo\_names):  
 offset = (i - 0.5) \* bar\_width  
 ax1.bar(x + offset, action\_counts[algo\_name], bar\_width,  
 label=algo\_name.upper(), color=algo\_colors[algo\_name], alpha=0.8)  
  
ax1.set\_xticks(x)  
ax1.set\_xticklabels(ACTION\_LABELS, rotation=15)  
ax1.set\_title("Action Counts (Absolute)", fontsize=13)  
ax1.set\_ylabel("Count")  
ax1.legend()  
ax1.grid(True, alpha=0.3, axis='y')  
  
# Percentage distribution  
for i, algo\_name in enumerate(algo\_names):  
 pcts = action\_counts[algo\_name] / action\_counts[algo\_name].sum() \* 100  
 offset = (i - 0.5) \* bar\_width  
 bars = ax2.bar(x + offset, pcts, bar\_width,  
 label=algo\_name.upper(), color=algo\_colors[algo\_name], alpha=0.8)  
 for bar, pct in zip(bars, pcts):  
 ax2.text(bar.get\_x() + bar.get\_width() / 2, bar.get\_height() + 0.5,  
 f'{pct:.1f}%', ha='center', va='bottom', fontsize=9)  
  
ax2.set\_xticks(x)  
ax2.set\_xticklabels(ACTION\_LABELS, rotation=15)  
ax2.set\_title("Action Distribution (Percentage)", fontsize=13)  
ax2.set\_ylabel("Percentage (%)")  
ax2.legend()  
ax2.grid(True, alpha=0.3, axis='y')  
  
fig.suptitle(f"DQN vs PPO: Action Distribution ({EVALUATION\_EPISODES \* len(SEED\_LIST)} episodes each)", fontsize=14)  
plt.tight\_layout()  
plt.show()



# Trajectory Plots: x-y paths of the lander  
  
fig, axes = plt.subplots(1, len(algo\_names), figsize=(8 \* len(algo\_names), 6), sharey=True)  
if len(algo\_names) == 1:  
 axes = [axes]  
  
traj\_colors = list(plt.colormaps["Set2"](range(8))) # type: ignore[arg-type]  
  
for ax, algo\_name in zip(axes, algo\_names):  
 for i, (x\_pos, y\_pos) in enumerate(trajectory\_data[algo\_name]):  
 ax.plot(x\_pos, y\_pos, color=traj\_colors[i], linewidth=1.5, alpha=0.8,  
 label=f"Episode {i+1}")  
 ax.scatter(x\_pos[0], y\_pos[0], color=traj\_colors[i], marker='o', s=60, zorder=5)  
 ax.scatter(x\_pos[-1], y\_pos[-1], color=traj\_colors[i], marker='x', s=80, zorder=5)  
  
 # Landing pad reference  
 ax.axhline(y=0, color='black', linestyle='-', linewidth=0.5, alpha=0.3)  
 ax.axvline(x=0, color='black', linestyle='-', linewidth=0.5, alpha=0.3)  
 ax.scatter(0, 0, color='red', marker='^', s=120, zorder=10, label='Landing Pad')  
  
 ax.set\_title(f"{algo\_name.upper()} Trajectories (seed {SEED\_LIST[0]})", fontsize=13)  
 ax.set\_xlabel("X Position")  
 ax.legend(fontsize=9)  
 ax.grid(True, alpha=0.3)  
  
axes[0].set\_ylabel("Y Position")  
fig.suptitle(f"Lander Trajectories: DQN vs PPO ({TRAJECTORY\_EPISODES} episodes each)", fontsize=14)  
plt.tight\_layout()  
plt.show()



# Trajectory Comparison: DQN vs PPO overlaid on one chart  
  
plt.figure(figsize=(10, 7))  
  
for algo\_name in algo\_names:  
 # Plot the first trajectory from each algorithm  
 x\_pos, y\_pos = trajectory\_data[algo\_name][0]  
 plt.plot(x\_pos, y\_pos, color=algo\_colors[algo\_name], linewidth=2, alpha=0.8,  
 label=f"{algo\_name.upper()}")  
 plt.scatter(x\_pos[0], y\_pos[0], color=algo\_colors[algo\_name], marker='o', s=80, zorder=5)  
 plt.scatter(x\_pos[-1], y\_pos[-1], color=algo\_colors[algo\_name], marker='x', s=100, zorder=5)  
  
plt.scatter(0, 0, color='red', marker='^', s=150, zorder=10, label='Landing Pad')  
plt.axhline(y=0, color='black', linestyle='-', linewidth=0.5, alpha=0.3)  
plt.axvline(x=0, color='black', linestyle='-', linewidth=0.5, alpha=0.3)  
  
plt.title(f"DQN vs PPO: Landing Trajectory Comparison (seed {SEED\_LIST[0]}, episode 1)", fontsize=14)  
plt.xlabel("X Position")  
plt.ylabel("Y Position")  
plt.legend(fontsize=11)  
plt.grid(True, alpha=0.3)  
plt.show()



## GIF Visualizations

# GIF Visualizations (one per algorithm per seed, best model)  
  
for algo\_name, algo\_class in ALGORITHM\_MAP.items():  
 output\_dir = os.path.join(NOTEBOOK\_DIR, "outputs\_" + algo\_name)  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 for seed in SEED\_LIST:  
 load\_path = best\_model\_paths.get(algo\_name, {}).get(seed)  
 if load\_path is None:  
 print(f"SKIPPING GIF for {algo\_name.upper()} seed {seed} — best model not found")  
 continue  
  
 print(f"Generating GIF for {algo\_name.upper()} seed {seed} (best model)...")  
  
 def make\_vis\_env(s=seed):  
 env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 env.reset(seed=s)  
 return env  
  
 vis\_model = algo\_class.load(load\_path, env=DummyVecEnv([make\_vis\_env]), device=DEVICE)  
  
 vis\_env = gym.make(GYMNASIUM\_MODEL, render\_mode="rgb\_array", enable\_wind=WIND\_ENABLED)  
 frames = []  
 obs, info = vis\_env.reset(seed=seed)  
 done = False  
  
 while not done:  
 action, \_ = vis\_model.predict(obs, deterministic=True)  
 obs, reward, terminated, truncated, info = vis\_env.step(action)  
 done = terminated or truncated  
 frames.append(vis\_env.render())  
  
 vis\_env.close()  
  
 gif\_path = os.path.join(output\_dir, f"{algo\_name}\_seed{seed}.gif")  
 imageio.mimsave(gif\_path, frames, fps=30)  
 print(f" Saved: {gif\_path}")  
 display(Image(filename=gif\_path))

Generating GIF for DQN seed 42 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_dqn/dqn\_seed42.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for DQN seed 123 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_dqn/dqn\_seed123.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for DQN seed 999 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_dqn/dqn\_seed999.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for PPO seed 42 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_ppo/ppo\_seed42.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for PPO seed 123 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_ppo/ppo\_seed123.gif  
  
  
  
<IPython.core.display.Image object>  
  
  
Generating GIF for PPO seed 999 (best model)...  
 Saved: /home/logus/env/iscte/taap\_p2/drafts/draft\_01/gymnasium/outputs\_ppo/ppo\_seed999.gif  
  
  
  
<IPython.core.display.Image object>

## Appendix: Experimental Setup

### Environment Details

| **Property** | **Value** |
| --- | --- |
| Environment | LunarLander-v3 (Gymnasium) |
| Observation Space | Box(8,) — continuous 8-dimensional vector |
| Action Space | Discrete(4) — do nothing, fire left, fire main, fire right |
| Solved Threshold | Mean reward >= 200 over 100 consecutive episodes |
| Wind | Disabled (enable\_wind=False) |

**Observation vector:** [x, y, vx, vy, angle, angular\_velocity, left\_leg\_contact, right\_leg\_contact]

**Reward structure:** - Moving toward the landing pad: positive - Moving away: negative - Crash: -100 - Successful landing: +100 - Each leg ground contact: +10 - Firing main engine: -0.3 per frame - Firing side engine: -0.03 per frame

**Termination rules:** - **Terminated (success):** The lander comes to rest on the ground with both legs in contact, near-zero velocity - **Terminated (crash):** The lander body contacts the ground, or the lander moves outside the viewport boundaries - **Truncated (timeout):** The episode exceeds 1000 timesteps without termination

# Environment inspection  
  
env\_tmp = gym.make(GYMNASIUM\_MODEL, enable\_wind=WIND\_ENABLED)  
print(f"Environment: {GYMNASIUM\_MODEL}")  
print(f"Observation space: {env\_tmp.observation\_space}")  
print(f"Action space: {env\_tmp.action\_space}")  
print(f"Wind enabled: {WIND\_ENABLED}")  
  
obs, info = env\_tmp.reset(seed=42)  
print(f"\nSample observation: {obs}")  
print(f"Observation labels: [x, y, vx, vy, angle, angular\_vel, left\_leg, right\_leg]")  
env\_tmp.close()

Environment: LunarLander-v3  
Observation space: Box([ -2.5 -2.5 -10. -10. -6.2831855 -10.  
 -0. -0. ], [ 2.5 2.5 10. 10. 6.2831855 10.  
 1. 1. ], (8,), float32)  
Action space: Discrete(4)  
Wind enabled: False  
  
Sample observation: [ 0.00229702 1.4181306 0.2326471 0.3204666 -0.00265488 -0.05269805  
 0. 0. ]  
Observation labels: [x, y, vx, vy, angle, angular\_vel, left\_leg, right\_leg]

# System and library versions  
  
import stable\_baselines3  
  
print(f"Python: {sys.version.split()[0]}")  
print(f"PyTorch: {torch.\_\_version\_\_}")  
print(f"Stable-Baselines3: {stable\_baselines3.\_\_version\_\_}")  
print(f"Gymnasium: {gym.\_\_version\_\_}")  
print(f"NumPy: {np.\_\_version\_\_}")  
print(f"Device: {DEVICE}")  
print(f"CUDA: {torch.version.cuda if torch.cuda.is\_available() else 'Not available'}")

Python: 3.12.3  
PyTorch: 2.10.0+cu130  
Stable-Baselines3: 2.7.1  
Gymnasium: 1.2.3  
NumPy: 2.4.2  
Device: cpu  
CUDA: 13.0