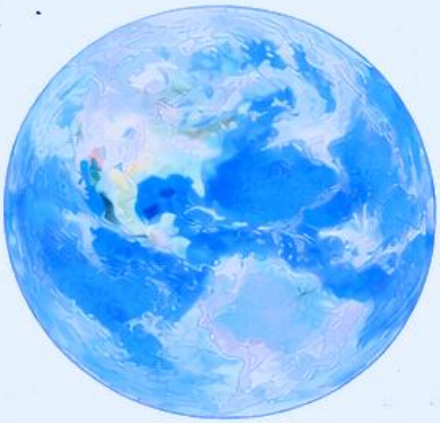
TAAP

INSTITUTO UNIVERSITÁRIO DE LISBOA



Advanced Topics in Deep Learning

MINI-PROJECT 02

DEEP REINFORCEMENT LEARNING
DQN AND PPO IN LUNARLANDER V3

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

Table of Contents

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, Callba
ckList
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.         -6.28318
55 -10.
  -0.         -0.       ], [ 2.5        2.5       10.        10.         6.283185
5 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_e
p{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=T
rue)
                    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:.0
f}%'
                    )
                    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).mea
n()
                    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 update
s
        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"checkpoi
nt_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(recen
t):.1f} | '
                            f'Min: {np.min(recent):.1f} | Max: {np.max(recent):.1
f} | '
                            f'Success: {(recent >= 200).sum() / len(recent) * 10
0:.0f}%'
                        )
                        if self._stats_handle is None:
                            self._stats_handle = display(stats_text, display_id=T
rue)
                        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(windo
w).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_re
wards)
            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 be
tter 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 fal
lback)
                    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)
```

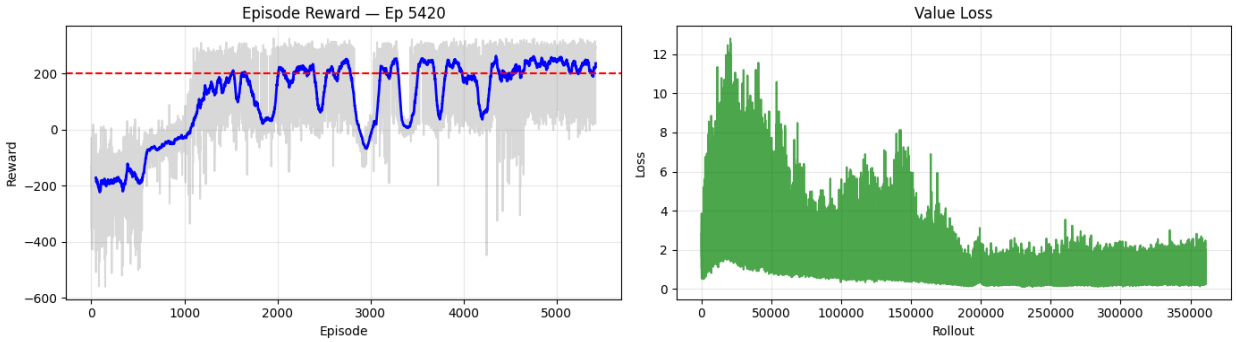
```
            else:
                self._eval_handle.update(eval_text)

            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_ti
mestamp)
        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_rewar
d:.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()
```

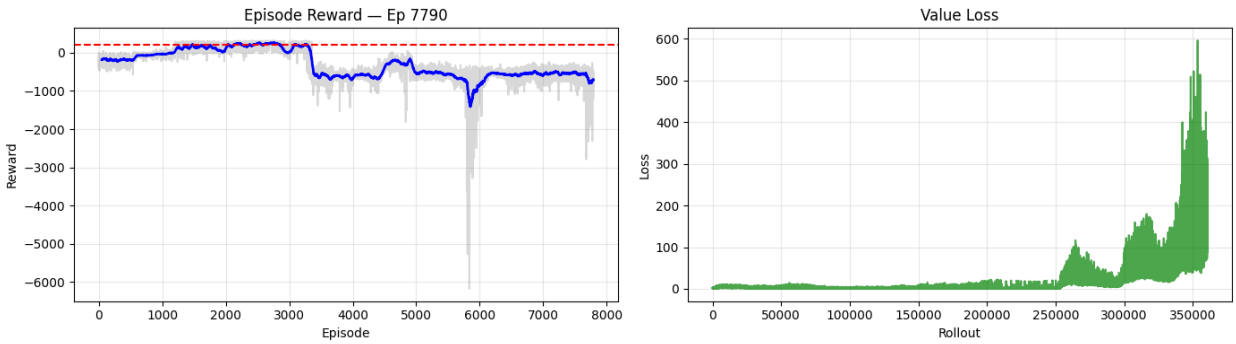
```
'Episode 5420 | Last 50 Ep — Mean: 236.2 | Std: 95.5 | Min: 15.8 | Max: 315.9 | S
uccess: 82%'
```




```
'[Checkpoint] Episode 5400 saved'
```

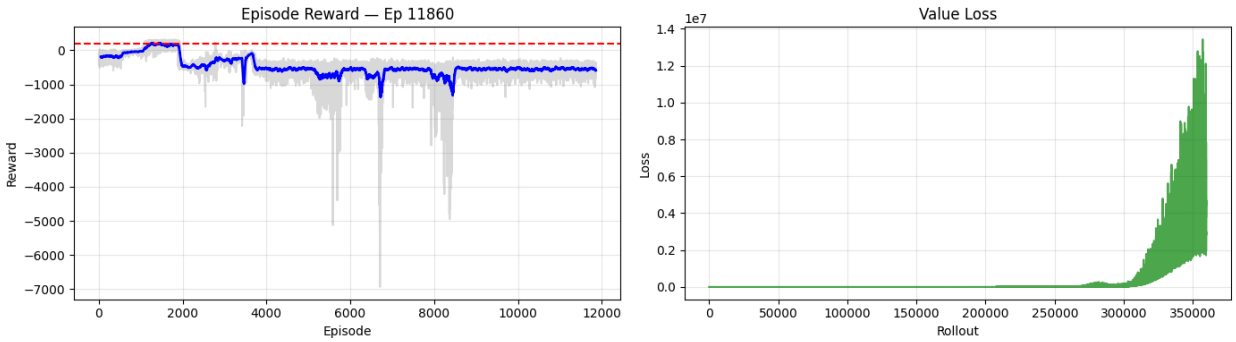
```
'Eval @ 1500000 steps | Reward: 276.50 +/- 16.66 [SOLVED] | Success: 100% | Score
 (mean-std): 259.85 | Best: 266.35'
Output()
```

```
'Episode 7790 | Last 50 Ep — Mean: -708.5 | Std: 349.2 | Min: -2311.8 | Max: -26
2.3 | Success: 0%'
```




```
Training time: 52.4 min (3146 s)
Final model:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_20_43_20/lab009_dqn_42
Best model:   /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_20_43_20/best_model
Checkpoints:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_20_43_20/checkpoints
Best model stats: Reward: 283.74 +/- 17.39 | Success: 100% | Score (mean-std): 26
6.35 | @ 1,425,000 steps
```

```
============================================================
DQN | Seed 123
============================================================



'[Checkpoint] Episode 7700 saved'



'Eval @ 1500000 steps | Reward: -926.29 +/- 368.67 | Success: 0% | Score (mean-st
d): -1294.96 | Best: 259.82'
Output()



'Episode 11860 | Last 50 Ep — Mean: -590.1 | Std: 162.5 | Min: -1082.2 | Max: -34
8.5 | Success: 0%'
```



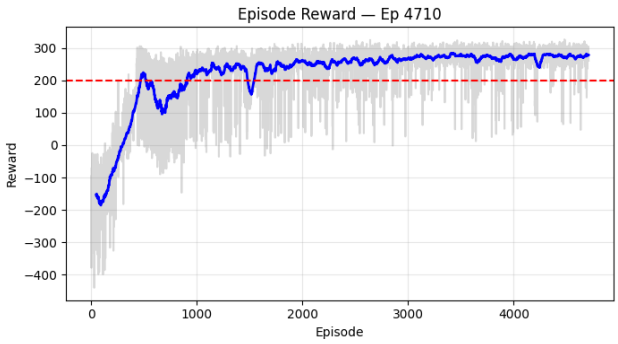

```
Training time: 51.5 min (3093 s)
Final model:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_21_35_46/lab009_dqn_123
Best model:   /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_21_35_46/best_model
Checkpoints:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_21_35_46/checkpoints
Best model stats: Reward: 275.61 +/- 15.80 | Success: 100% | Score (mean-std): 25
9.82 | @ 875,000 steps

============================================================
DQN | Seed 999
============================================================
```

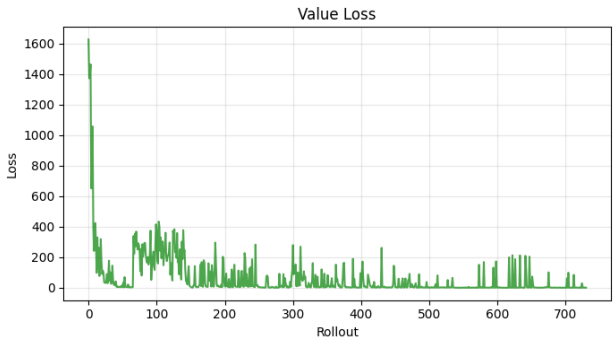
```
'[Checkpoint] Episode 11800 saved'
```

```
'Eval @ 1500000 steps | Reward: -593.75 +/- 172.38 | Success: 0% | Score (mean-st
d): -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/../../../mo
dels/dqn/2026-02-21_22_27_19/lab009_dqn_999
Best model:   /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_22_27_19/best_model
Checkpoints:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/dqn/2026-02-21_22_27_19/checkpoints
Best model stats: Reward: 266.25 +/- 22.38 | Success: 100% | Score (mean-std): 24
3.87 | @ 550,000 steps

DQN: All 3 seeds trained.

============================================================
PPO | Seed 42
============================================================
```

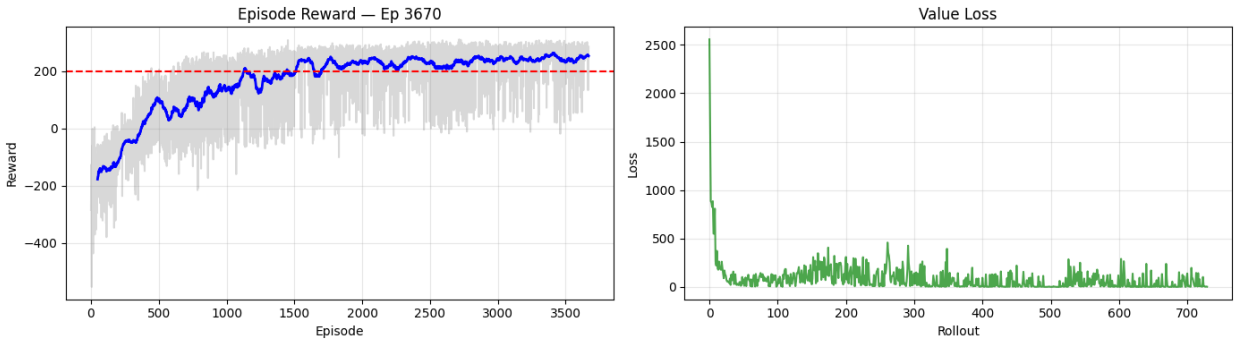
```
'Episode 4710 | Last 50 Ep — Mean: 278.3 | Std: 31.3 | Min: 146.7 | Max: 320.5 |
Success: 96%'
```






```
'[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/../../../mo
dels/ppo/2026-02-21_23_17_45/lab009_ppo_42
Best model:   /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/ppo/2026-02-21_23_17_45/best_model
Checkpoints:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
dels/ppo/2026-02-21_23_17_45/checkpoints
Best model stats: Reward: 289.50 +/- 21.48 | Success: 100% | Score (mean-std): 26
8.02 | @ 1,500,000 steps

============================================================
PPO | Seed 123
============================================================



'Episode 3670 | Last 50 Ep — Mean: 252.4 | Std: 46.9 | Min: 56.7 | Max: 301.1 | S
uccess: 90%'
```

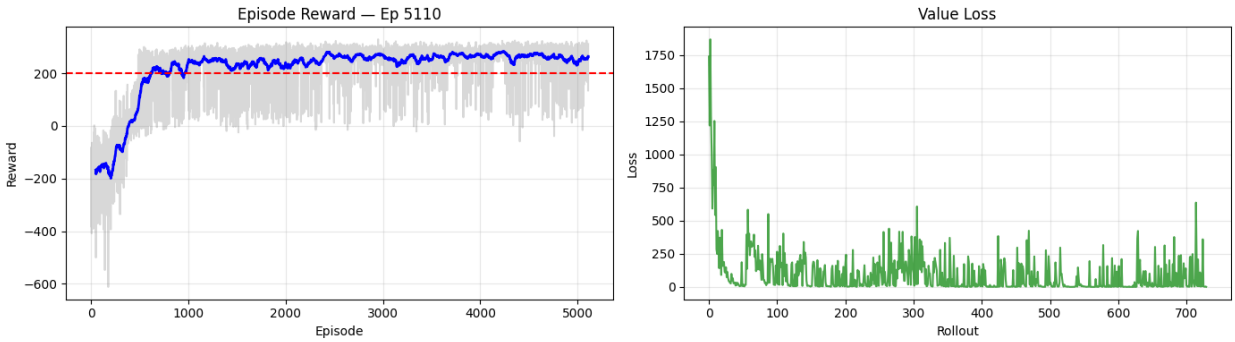



```
'[Checkpoint] Episode 3600 saved'



'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'
```

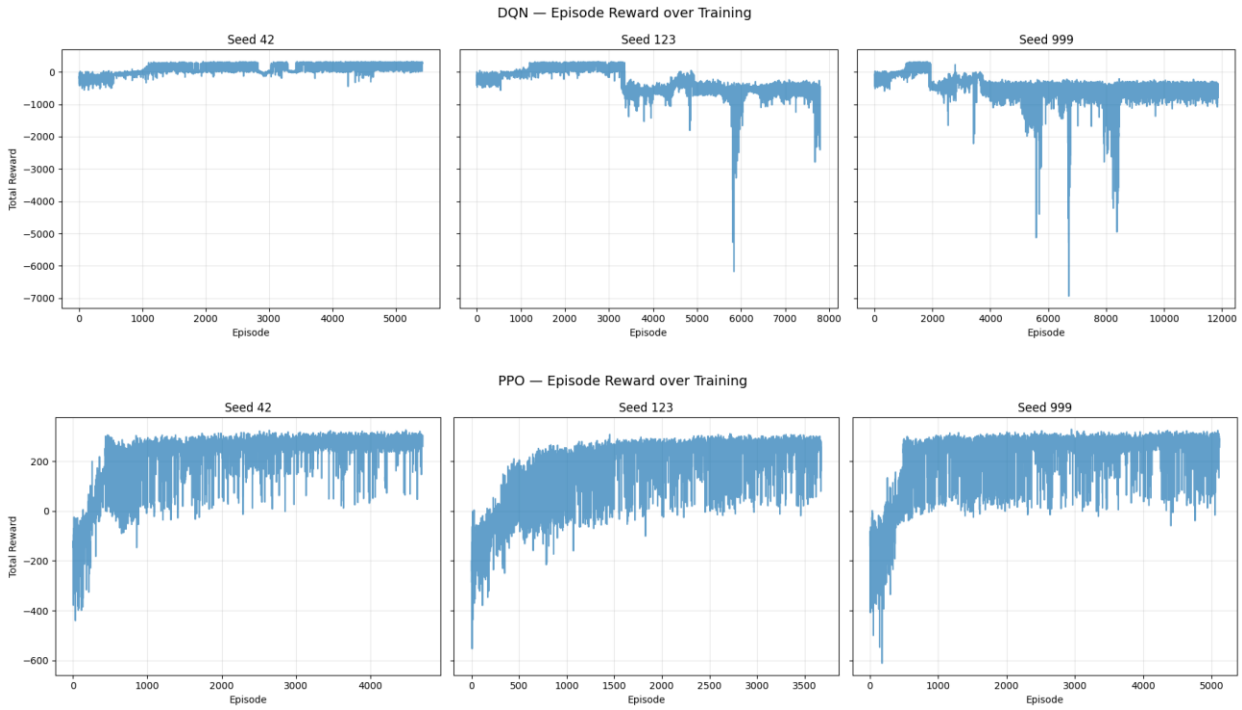
```
'Eval @ 1500000 steps | Reward: 271.94 +/- 35.27 [SOLVED] | Success: 95% | Score (mean-std): 236.66 | Best: 266.00'
Training time: 20.6 min (1235 s)
Final model:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../models/ppo/2026-02-22_00_03_53/lab009_ppo_999
Best model:   /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../models/ppo/2026-02-22_00_03_53/best_model
Checkpoints:  /home/logus/env/iscte/taap_p2/drafts/draft_01/gymnasium/../../../mo
```

```
dels/ppo/2026-02-22_00_03_53/checkpoints
Best model stats: Reward: 285.68 +/- 19.69 | Success: 100% | Score (mean-std): 26
6.00 | @ 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))
```

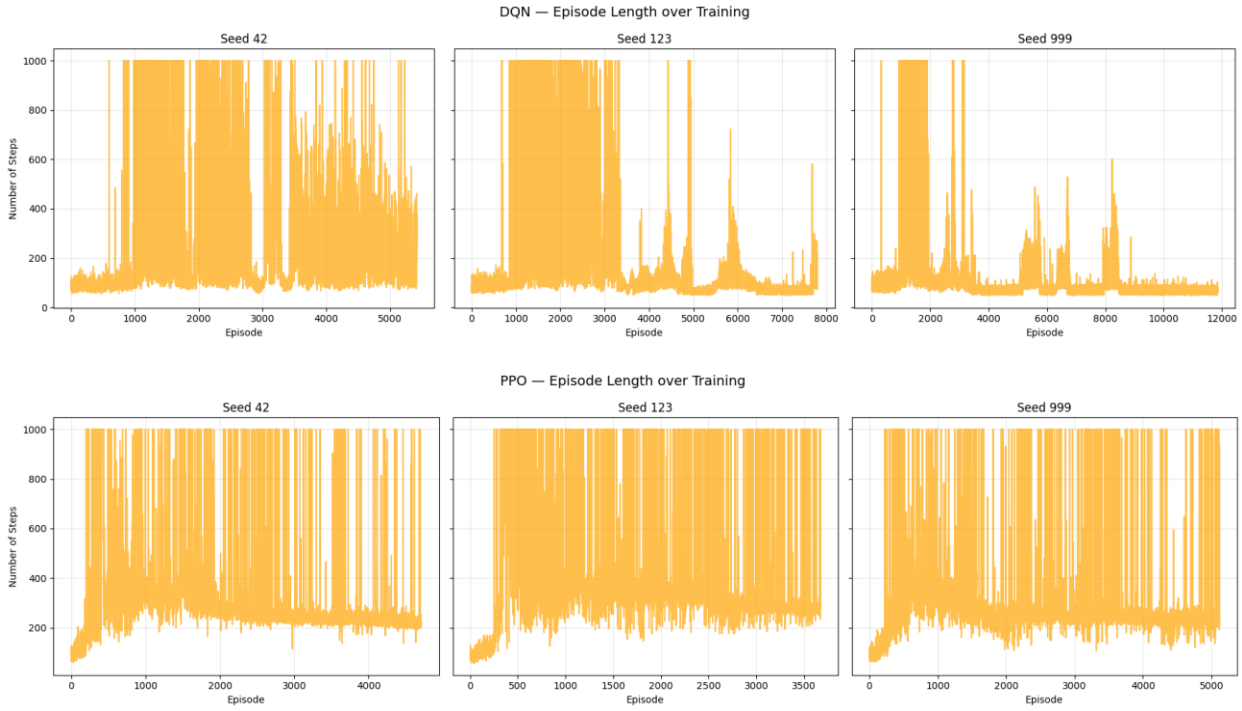
```
*** TRAINING TIME SUMMARY ***
Timesteps per seed: 1,500,000 | Device: cpu

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 cal
l, 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 (upd
ates/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, call
ed 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", font
size=14)
```

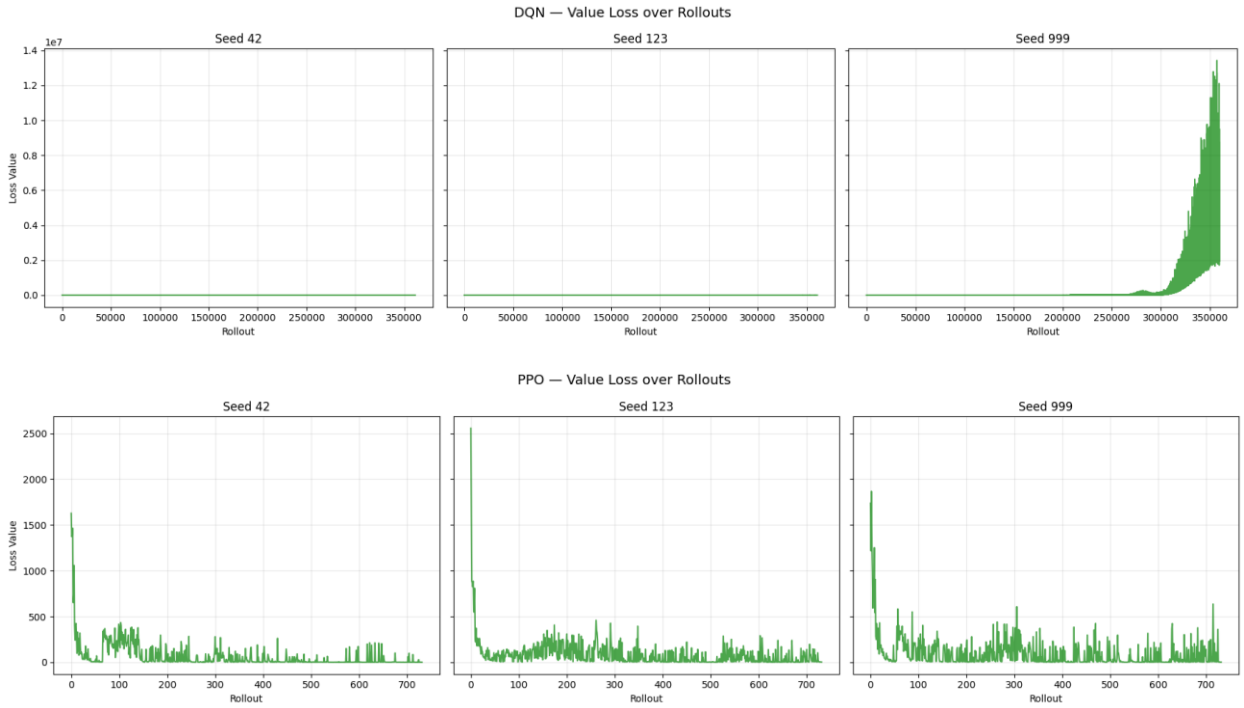
```
    plt.tight_layout()
    plt.show()
```



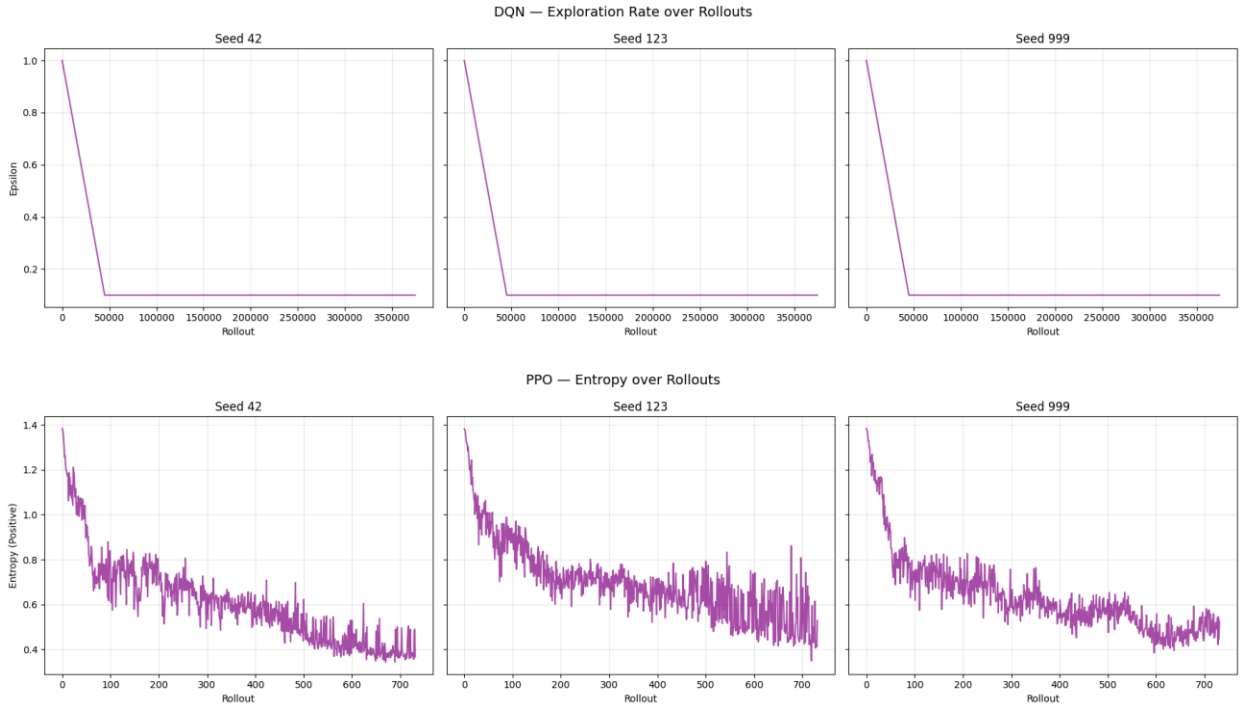

```
# Per-Algorithm, Per-Seed: Episode Length over Training

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, col
or="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", font
size=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()
```

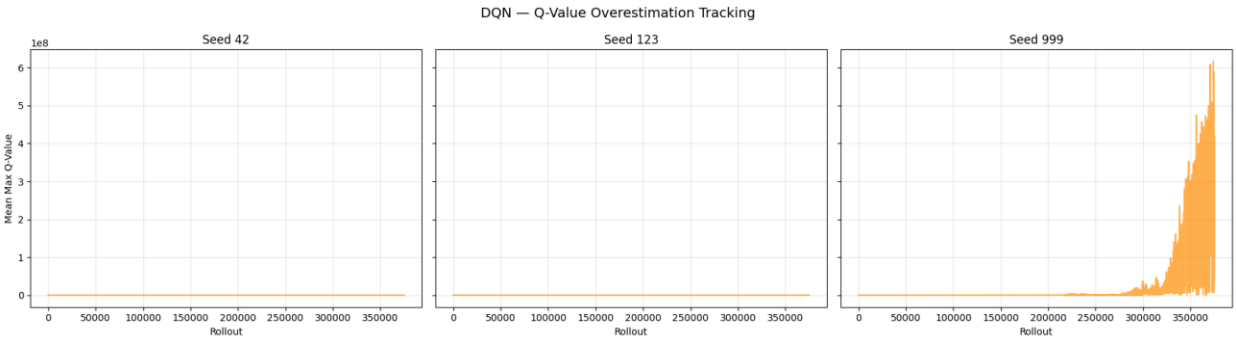



```
# Per-Algorithm, Per-Seed: Entropy / Exploration Rate over Rollouts

entropy_labels = {"dqn": ("Epsilon", "Exploration Rate"), "ppo": ("Entropy (Posit
ive)", "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="purp
le")
        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", font
size=14)
    plt.tight_layout()
    plt.show()
```



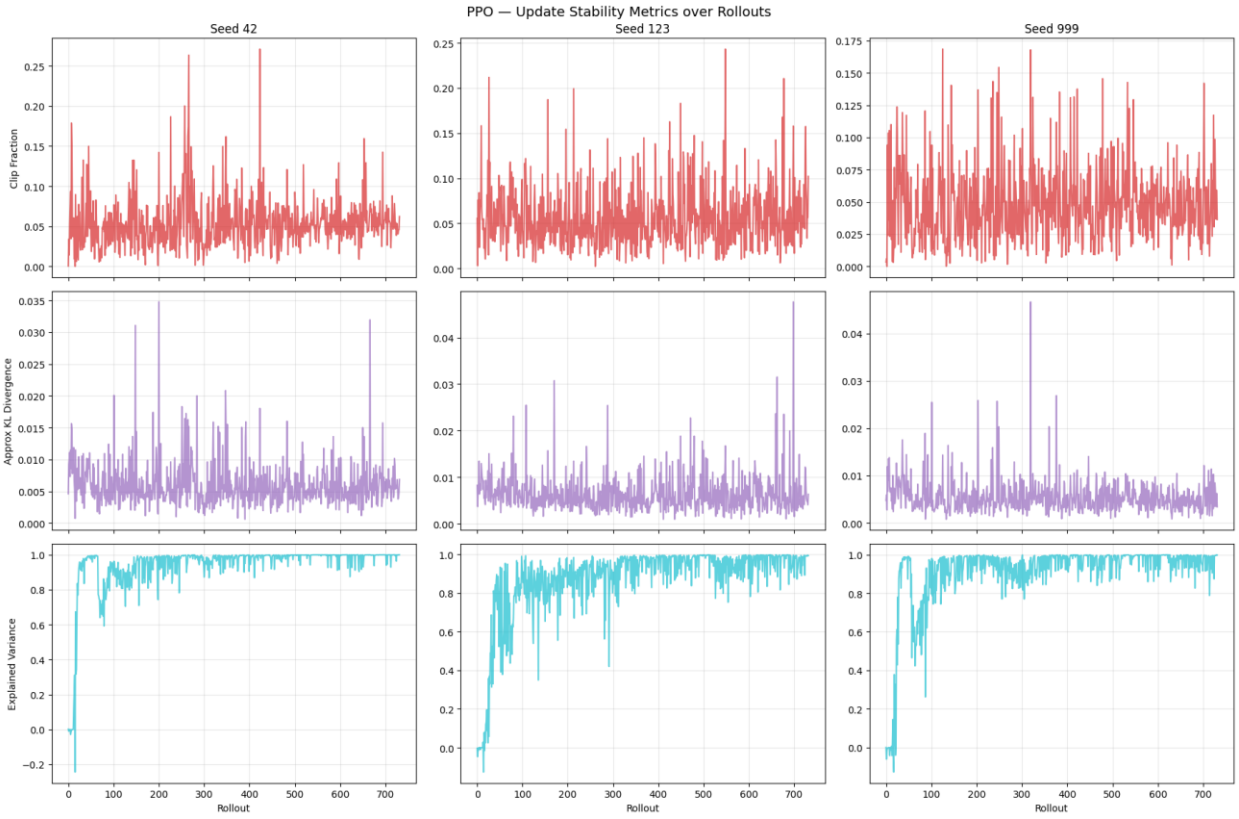

```
# PPO Only: Policy Loss over Rollouts

fig, axes = plt.subplots(1, len(SEED_LIST), figsize=(6 * len(SEED_LIST), 5), shar
ey=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), shar
ey=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 o
verestimation.")
print("A stable or slowly growing curve suggests the target network is controllin
g overestimation.")
```



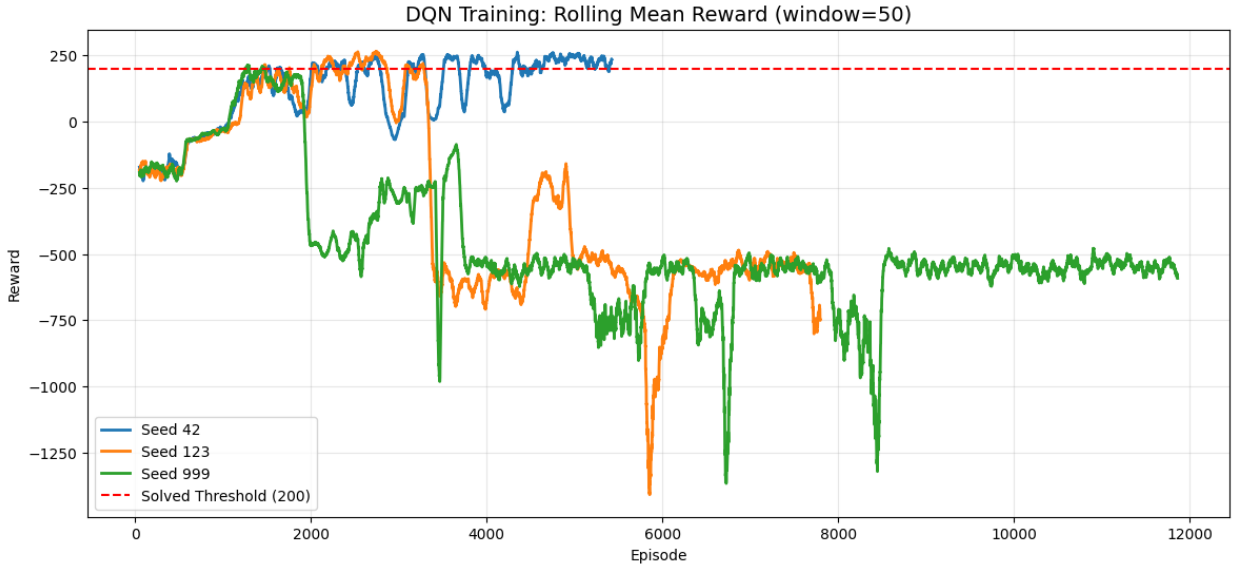

```
Note: Steadily rising Q-values that diverge from actual returns indicate overesti
mation.
A stable or slowly growing curve suggests the target network is controlling overe
stimation.
# PPO Update Stability: Clip Fraction, Approx KL, Explained Variance

fig, axes = plt.subplots(3, len(SEED_LIST), figsize=(6 * len(SEED_LIST), 12), sha
rex=True)
if len(SEED_LIST) == 1:
    axes = axes.reshape(3, 1)
```

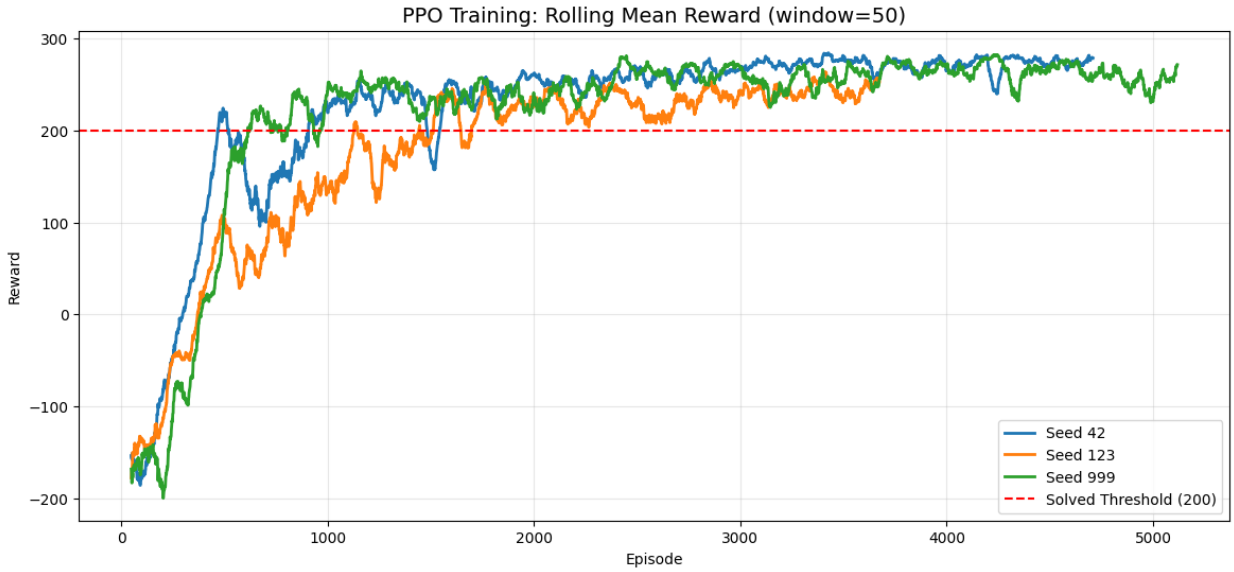
```
metrics = [
    ("clip_fraction", "Clip Fraction", "tab:red",
     "Fraction of policy updates clipped by PPO. High values suggest the policy i
s changing too fast."),
    ("approx_kl", "Approx KL Divergence", "tab:purple",
     "KL divergence between old and new policy. Spikes indicate large policy shif
ts."),
    ("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 t
he 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 = perfec
t, 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)", f
ontsize=14)
    plt.xlabel("Episode")
    plt.ylabel("Reward")
```

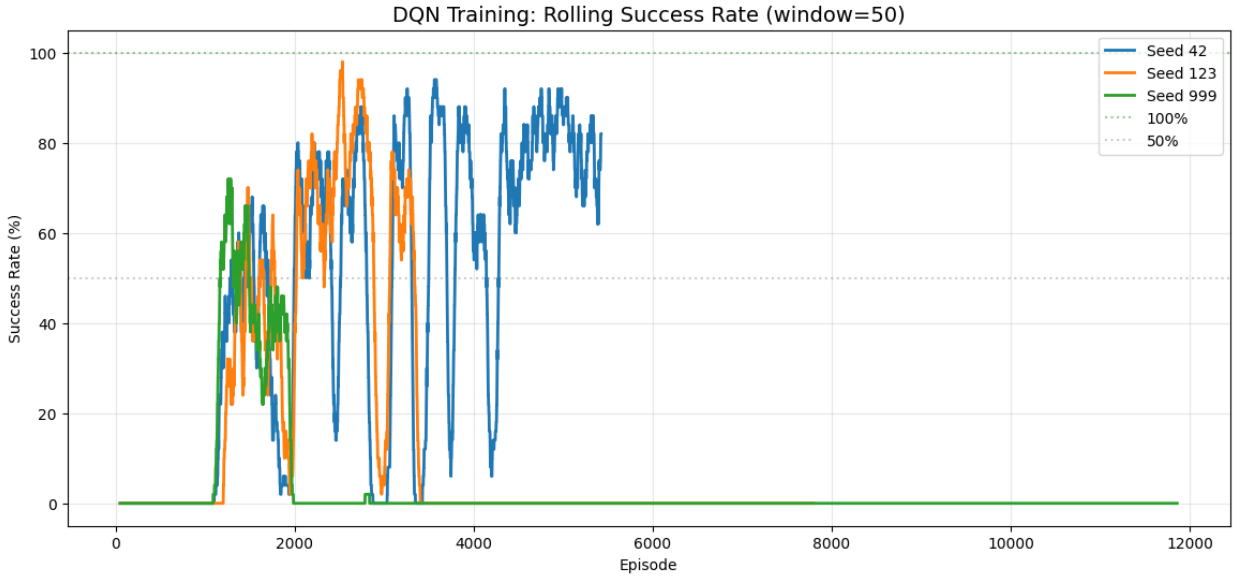
```
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()
```

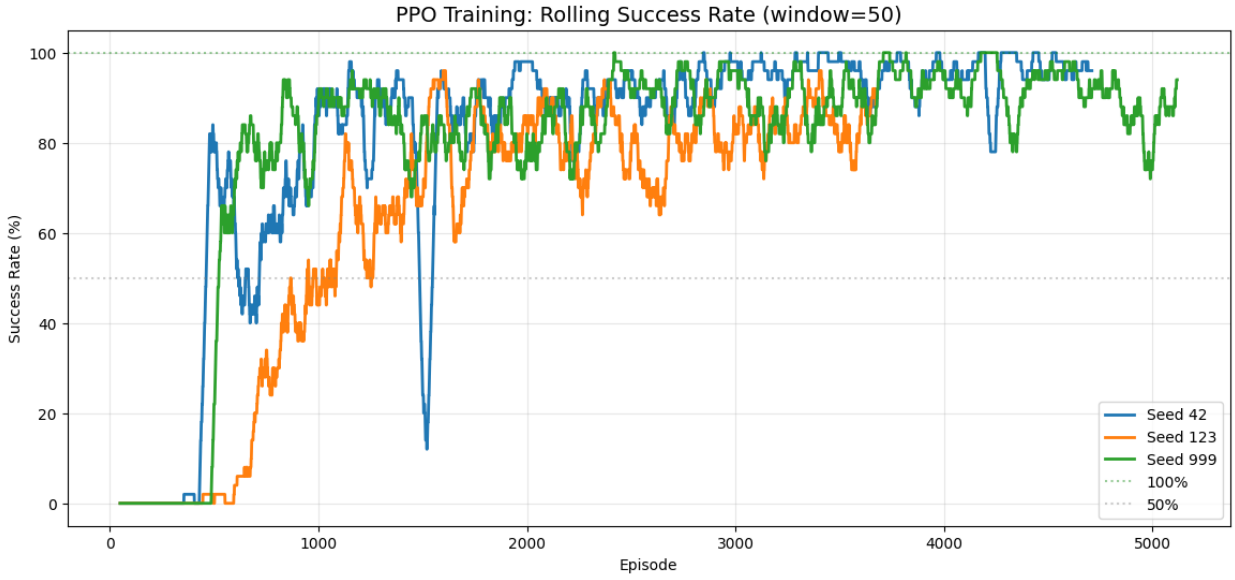





```
# Per-Algorithm: Rolling Success Rate over Training (window=50)

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}")
```

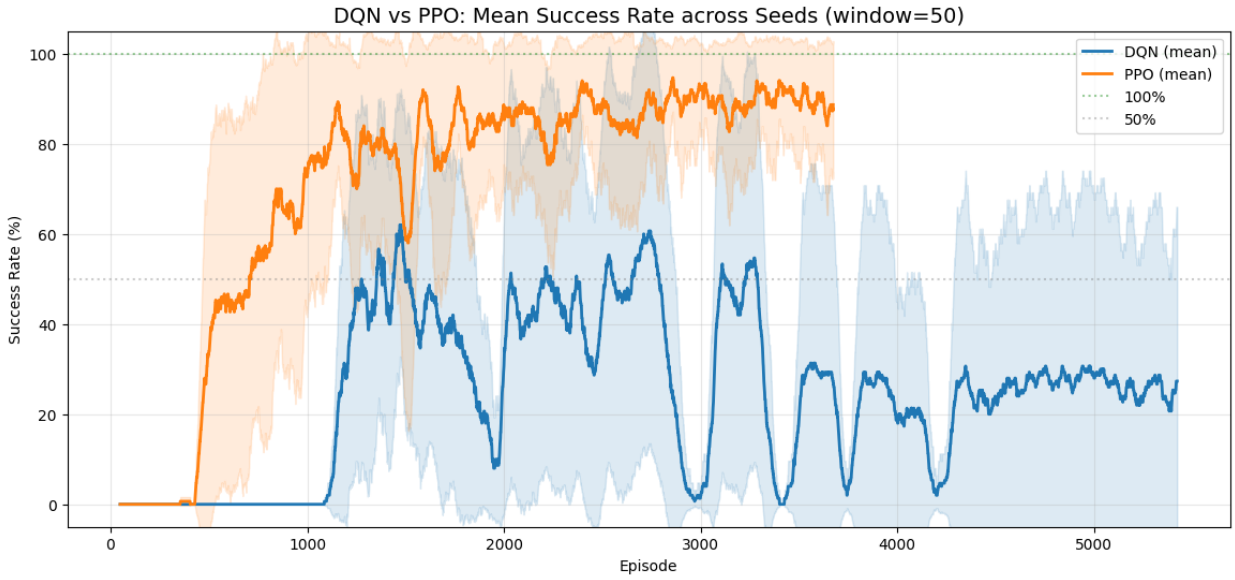
```
    plt.axhline(y=100, color='green', linestyle=':', alpha=0.4, label='100%')
    plt.axhline(y=50, color='gray', linestyle=':', alpha=0.4, label='50%')
    plt.title(f"{algo_name.upper()} Training: Rolling Success Rate (window=50)",
fontsize=14)
    plt.xlabel("Episode")
    plt.ylabel("Success Rate (%)")
    plt.ylim(-5, 105)
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()
```



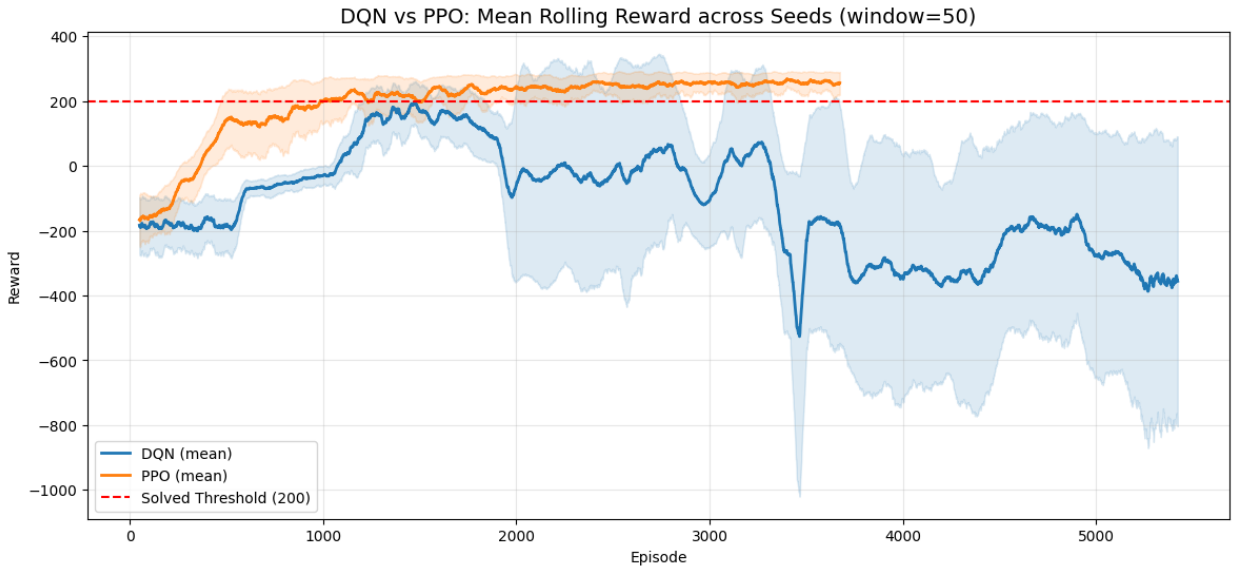



```
# Cross-Algorithm Comparison: Success Rate over Training (averaged across seeds)

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 SE
ED_LIST)
    all_rewards = np.array([training_results[algo_name][s].episode_rewards[:min_l
en] 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_suc
cess,
                     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()
```



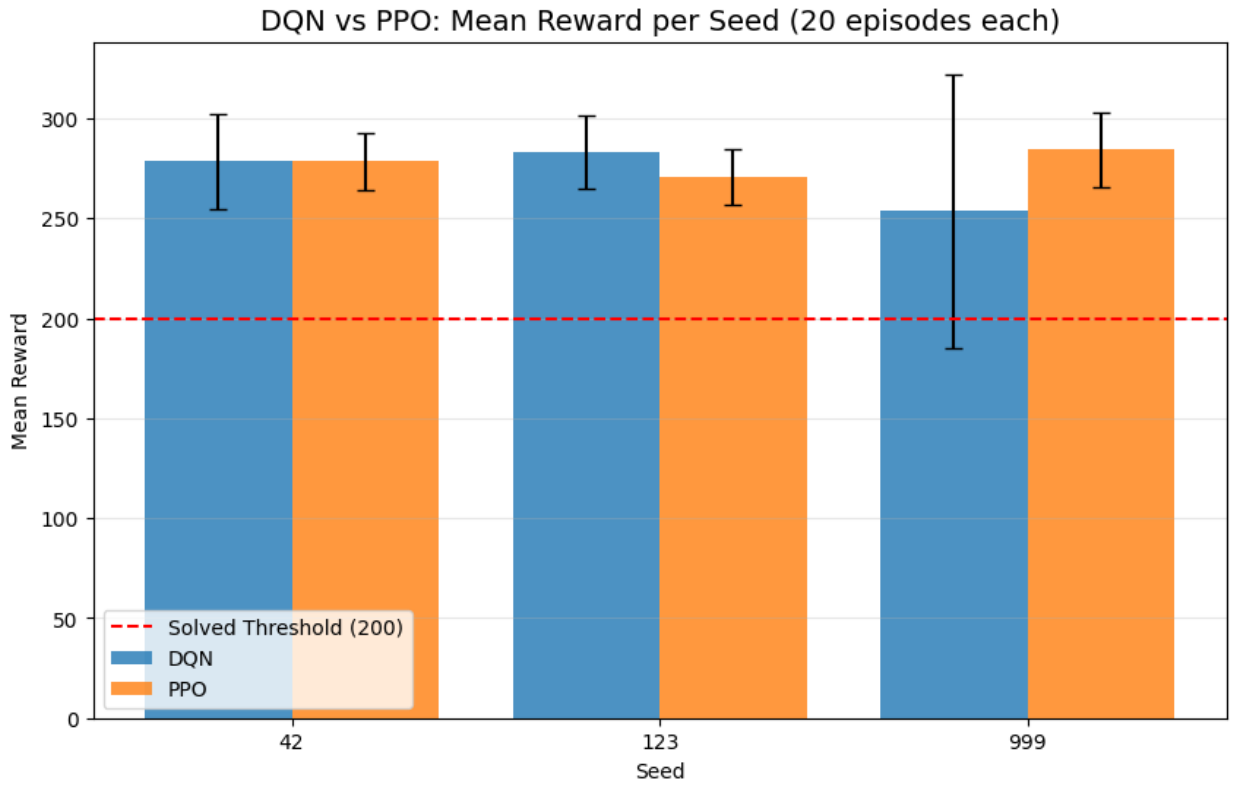
```
# Cross-Algorithm Comparison: Rolling Reward (averaged across seeds)

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 SE
ED_LIST)
    all_rewards = np.array([training_results[algo_name][s].episode_rewards[:min_l
en] 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, l
abel=f"{algo_name.upper()} (mean)")
    plt.fill_between(episodes, mean_rewards - std_rewards, mean_rewards + std_rew
ards,
                     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=1
4)
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```



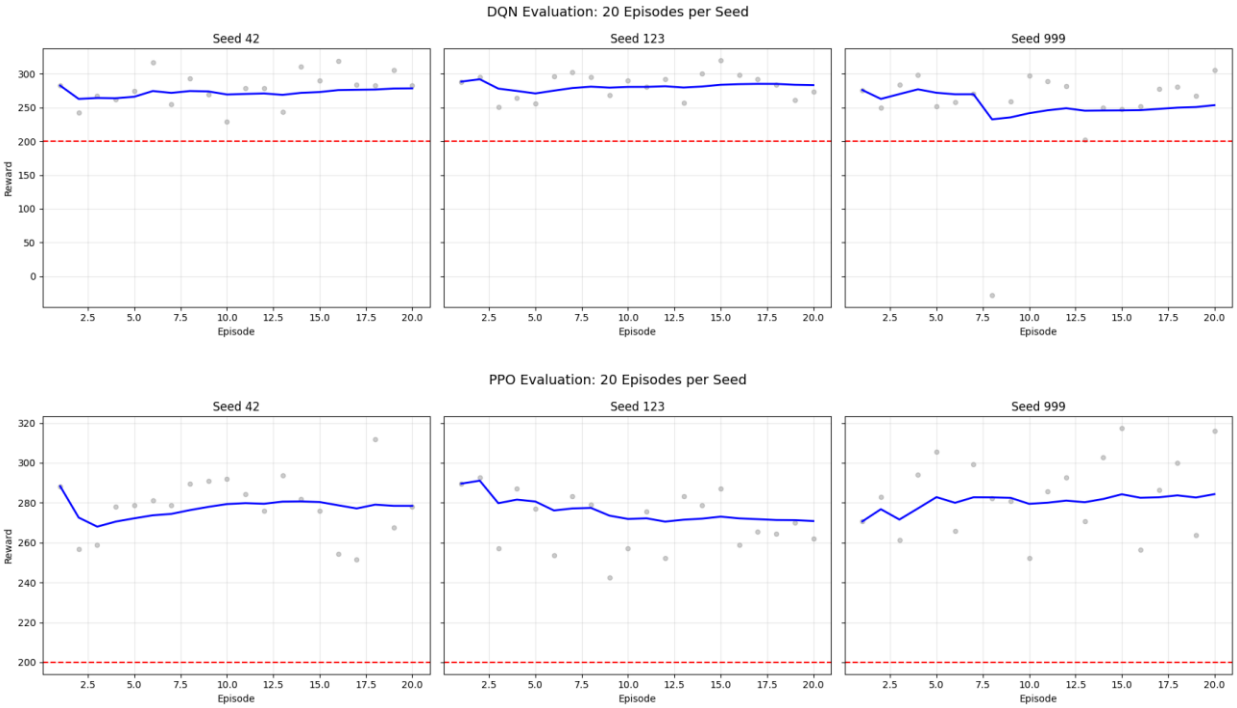
```
# Evaluation: deterministic episodes per algorithm per seed (best model)

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 eac
h)", 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 mode
l)...")

        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]), d
evice=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_TIMES
TEPS)})
    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                                                  Val
ue
                 policy                                                MlpPoli
cy
          learning_rate <function linear_schedule.<locals>.func at 0x7058383c356
0>
         learning_starts                                                 500
00
```

```
             buffer_size                                                 7500
00
              batch_size                                                  1
28
                   gamma                                                  0.
99
    exploration_fraction                                                  0.
12
   exploration_final_eps
0.1
  target_update_interval                                                  2
50
              train_freq
 4
          gradient_steps
 4
           policy_kwargs                                 {'net_arch': [256, 25
6]}
                  device                                                  c
pu
         total_timesteps                                               15000
00
                  device                                                  c
pu
                  policy                                              MlpPoli
cy
checkpoint_freq_episodes                                                  1
00
     eval_freq_timesteps                                                250
00
         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.zi
p":
                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 plo
```

```
ts

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] = {}
```

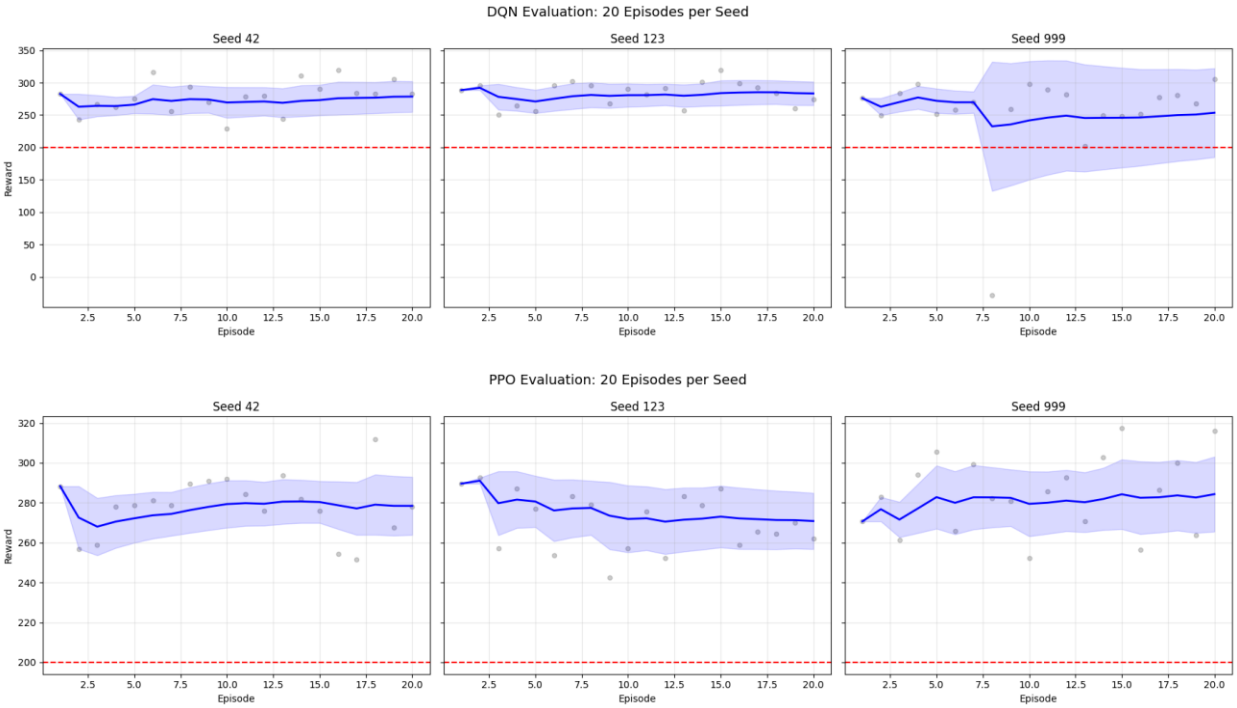
```
    for seed in SEED_LIST:
        load_path = best_model_paths.get(algo_name, {}).get(seed)
        if load_path is None:
            print(f"SKIPPING {algo_name.upper()} seed {seed} — best model not fou
nd")
            continue

        print(f"Loading and evaluating {algo_name.upper()} seed {seed} (best mode
l)...")

        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=DE
VICE)

        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.
```

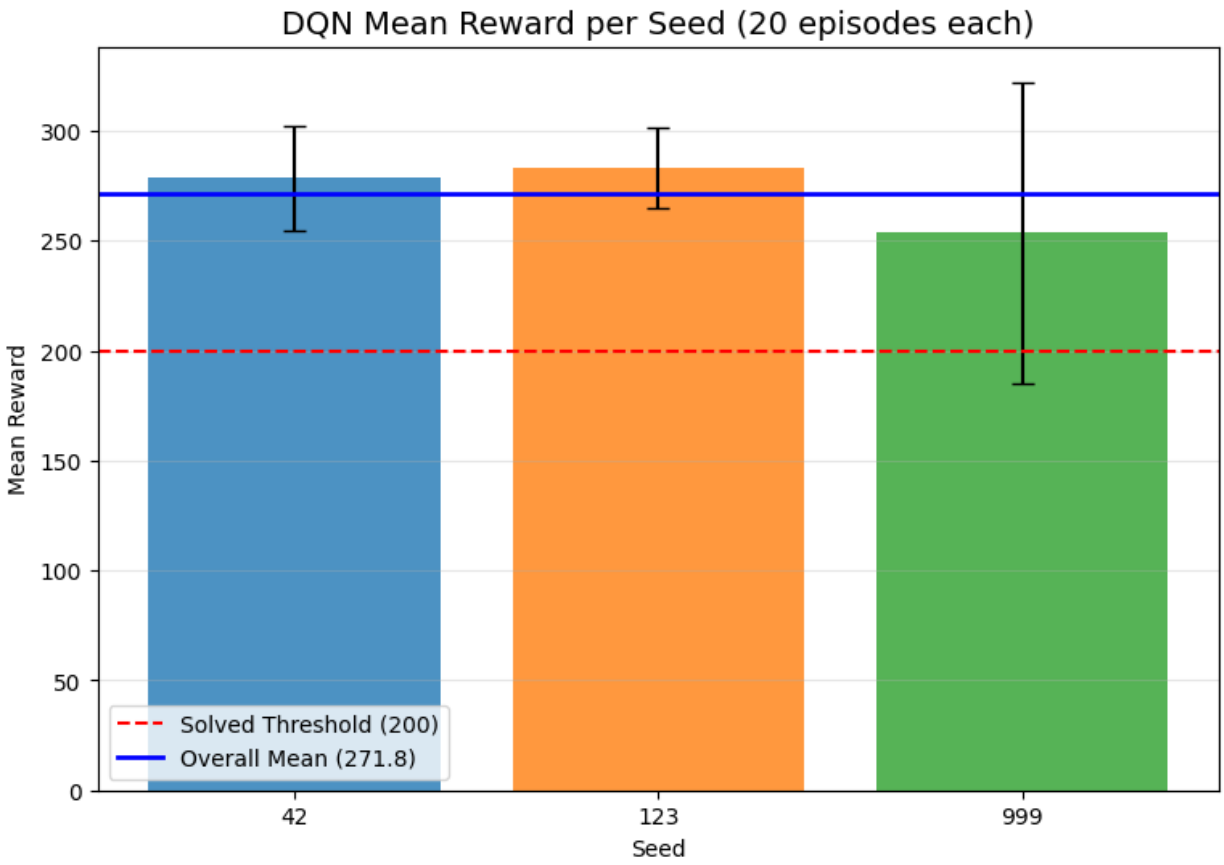
Per-Algorithm Results

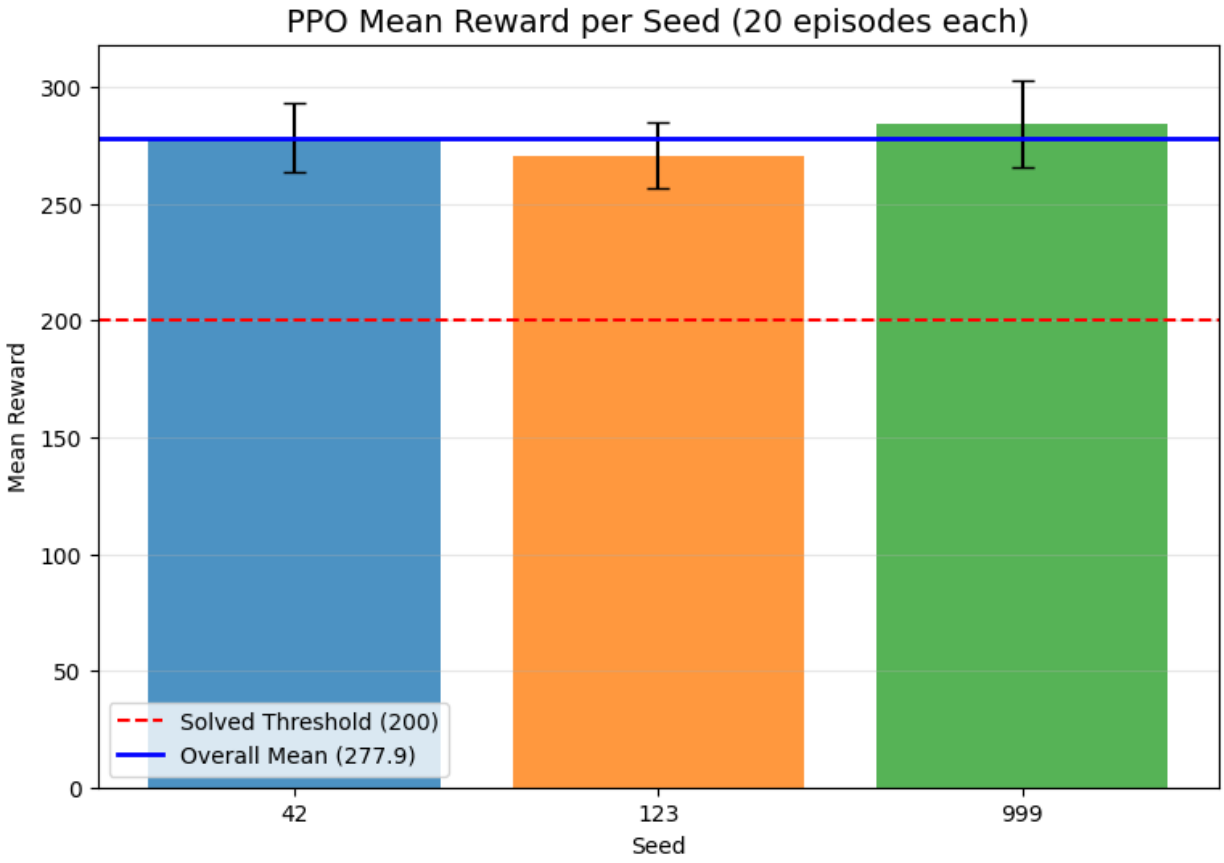
```
# Per-Algorithm: Evaluation Summary Tables

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='Episo
de Reward')
        ax.plot(episodes, running_mean, color='blue', linewidth=2, label='Running
 Mean')
        ax.fill_between(episodes, running_mean - running_std, running_mean + runn
ing_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()
```

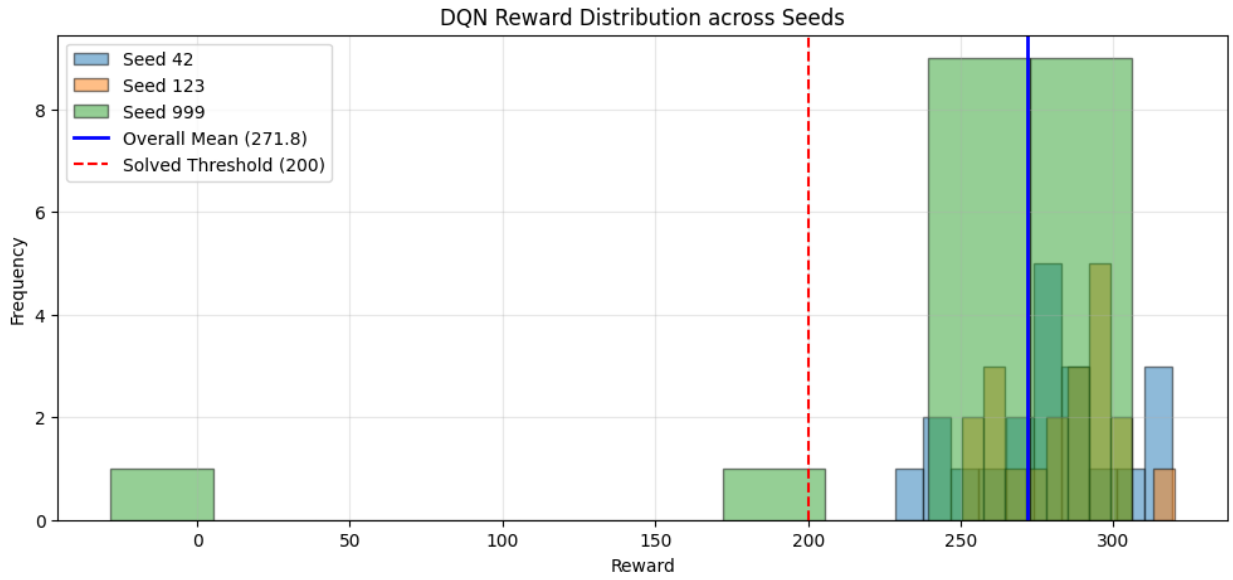


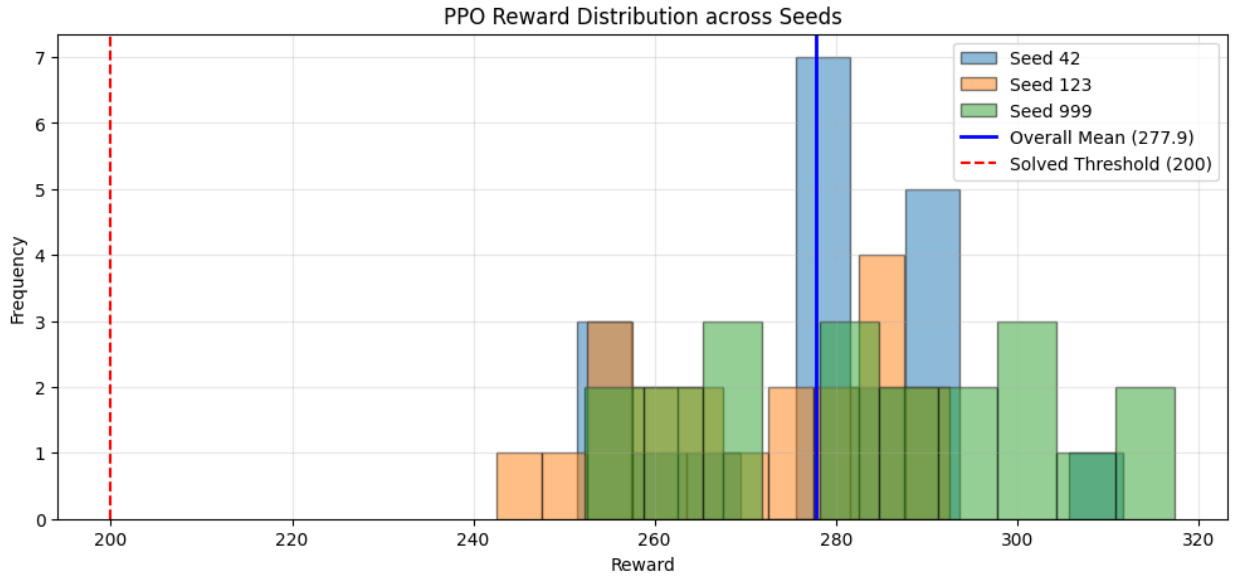

```
# Per-Algorithm: Evaluation Bar Chart (mean reward per seed with error bars)

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_LIS
T)], 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} e
pisodes each)", fontsize=14)
    plt.xlabel("Seed")
    plt.ylabel("Mean Reward")
    plt.legend()
    plt.grid(True, alpha=0.3, axis='y')
    plt.show()
```


DQN Mean Reward per Seed (20 episodes each)
Mean Reward
300
250
200
150
100
50
0
Solved Threshold (200)
Overall Mean (271.8)
42
123
999
Seed


PPO Mean Reward per Seed (20 episodes each)
Mean Reward
300
250
200
150
100
50
0
Solved Threshold (200)
Overall Mean (277.9)
42
123
999
Seed

```
# 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()
```



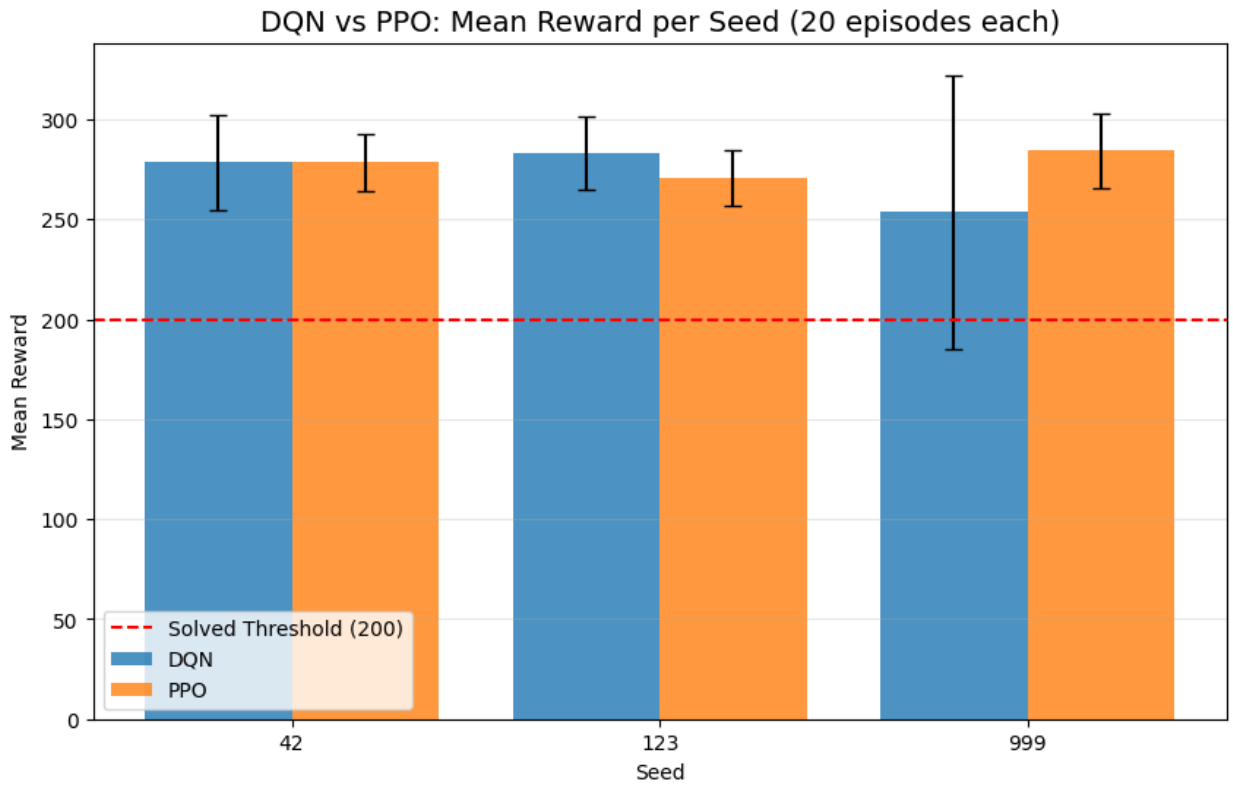


Cross-Algorithm Comparison

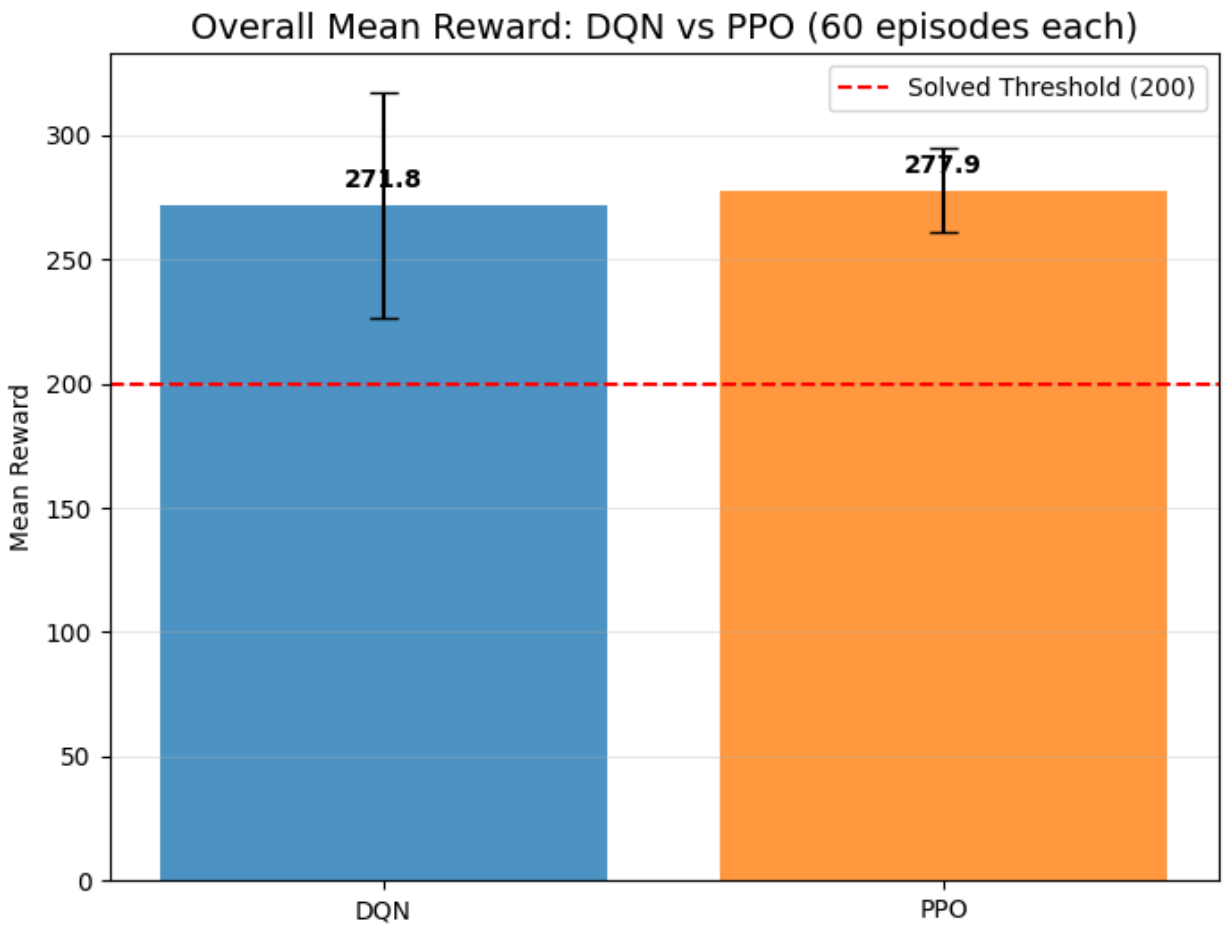
```
# Cross-Algorithm: Combined Summary Table

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%
```

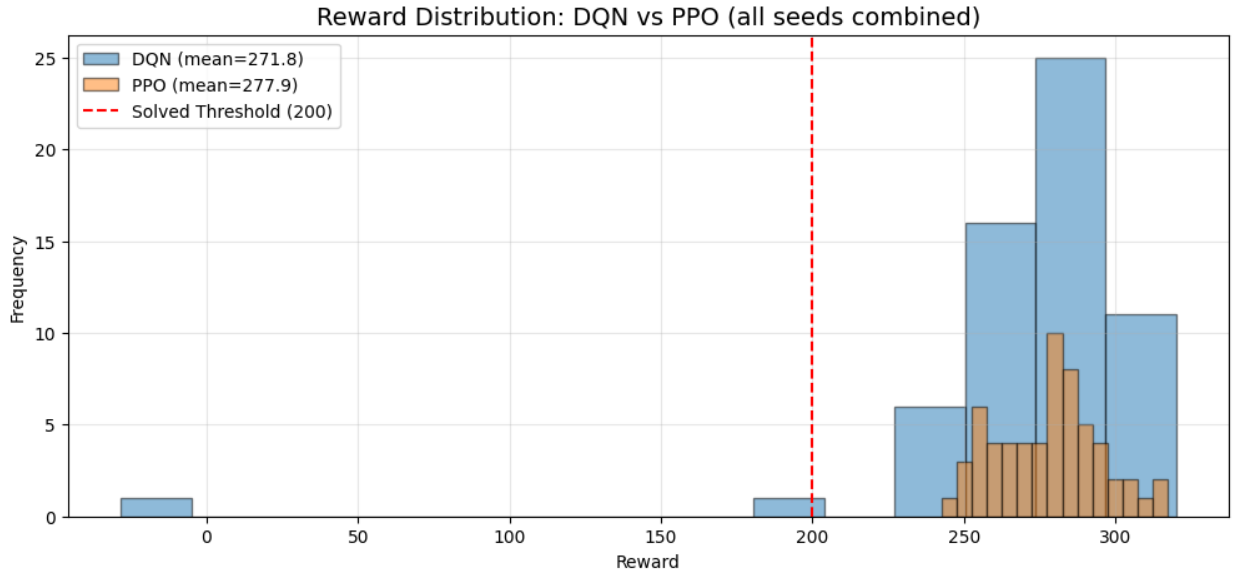
```
# Cross-Algorithm: Grouped Bar Chart (DQN vs PPO per seed)

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 eac
h)", 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_LIS
T)} episodes each)", fontsize=14)
plt.ylabel("Mean Reward")
plt.legend()
plt.grid(True, alpha=0.3, axis='y')
plt.show()
```

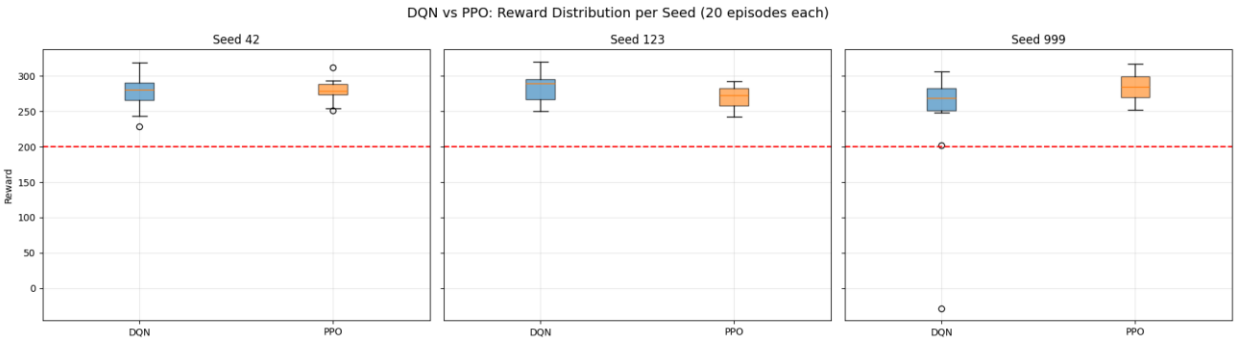



```
# Cross-Algorithm: Reward Distribution Comparison (overlaid histograms)

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), shar
ey=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=Tr
ue)
    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} ep
isodes 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; suppo
rt for the old name will be dropped in 3.11.
  bp = ax.boxplot(data, labels=[a.upper() for a in algo_names], patch_artist=Tru
e)
/tmp/ipykernel_86514/3208474440.py:9: MatplotlibDeprecationWarning: The 'labels'
parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; suppo
```

```
rt for the old name will be dropped in 3.11.
  bp = ax.boxplot(data, labels=[a.upper() for a in algo_names], patch_artist=Tru
e)
/tmp/ipykernel_86514/3208474440.py:9: MatplotlibDeprecationWarning: The 'labels'
parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; suppo
rt for the old name will be dropped in 3.11.
  bp = ax.boxplot(data, labels=[a.upper() for a in algo_names], patch_artist=Tru
e)
```



Statistical Significance

```python
# 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-z
ero
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} ep
isodes 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% o
r 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 Signifi
cant (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)
```

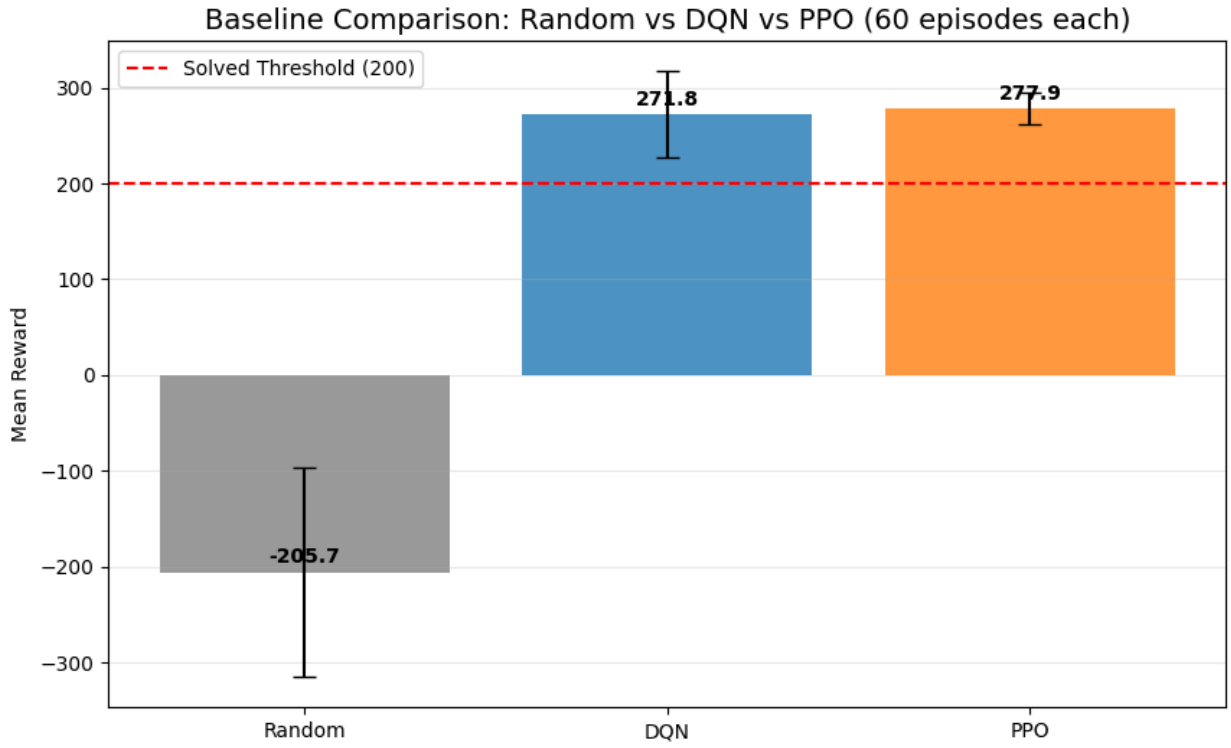
```
        total_reward = 0.0
        done = False

        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_LIS
T]) 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:.1
f}%"
    }
]
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_na
mes]
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%
```

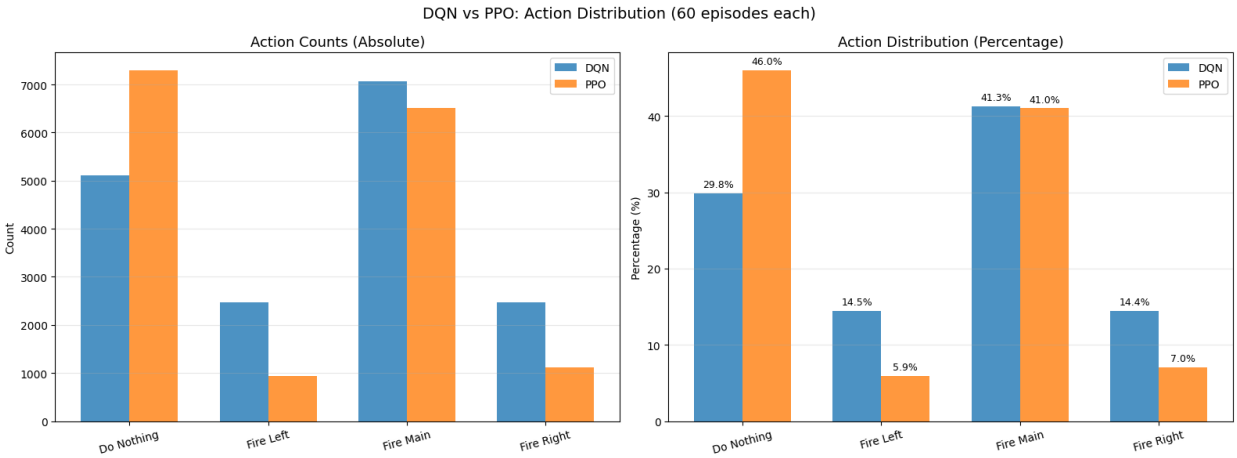


Agent Behavior Analysis

```
# 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 TRA
JECTORY_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
```

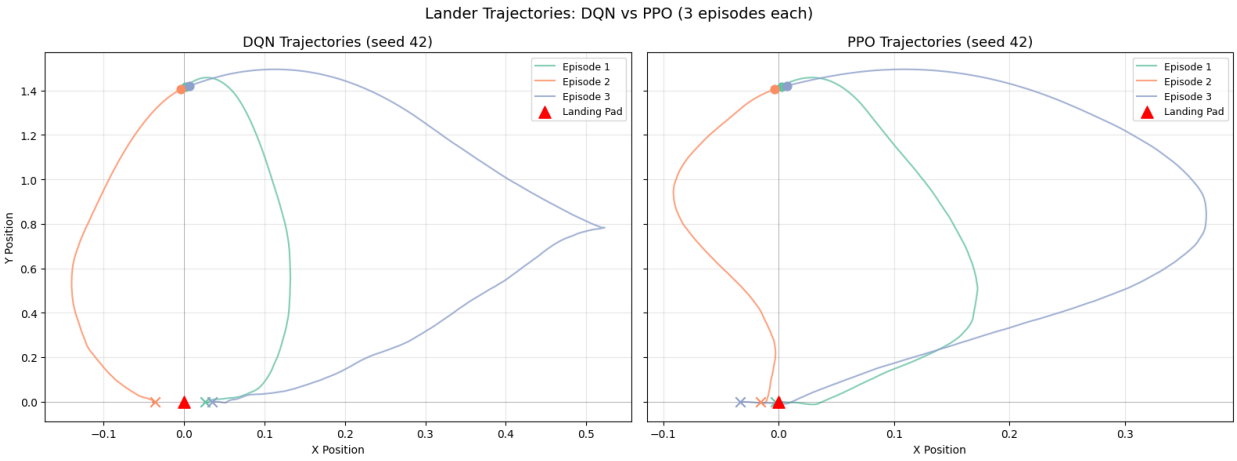
```
        model = algo_class.load(load_path, env=DummyVecEnv([make_env]), device=DE
VICE)
        env = gym.make(GYMNASIUM_MODEL, enable_wind=WIND_ENABLED)

        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 f
irst seed
            if seed == SEED_LIST[0] and ep < TRAJECTORY_EPISODES:
                trajectory_data[algo_name].append((np.array(x_pos), np.array(y_po
s)))

        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_L
IST)} episodes each)", fontsize=14)
plt.tight_layout()
plt.show()
```

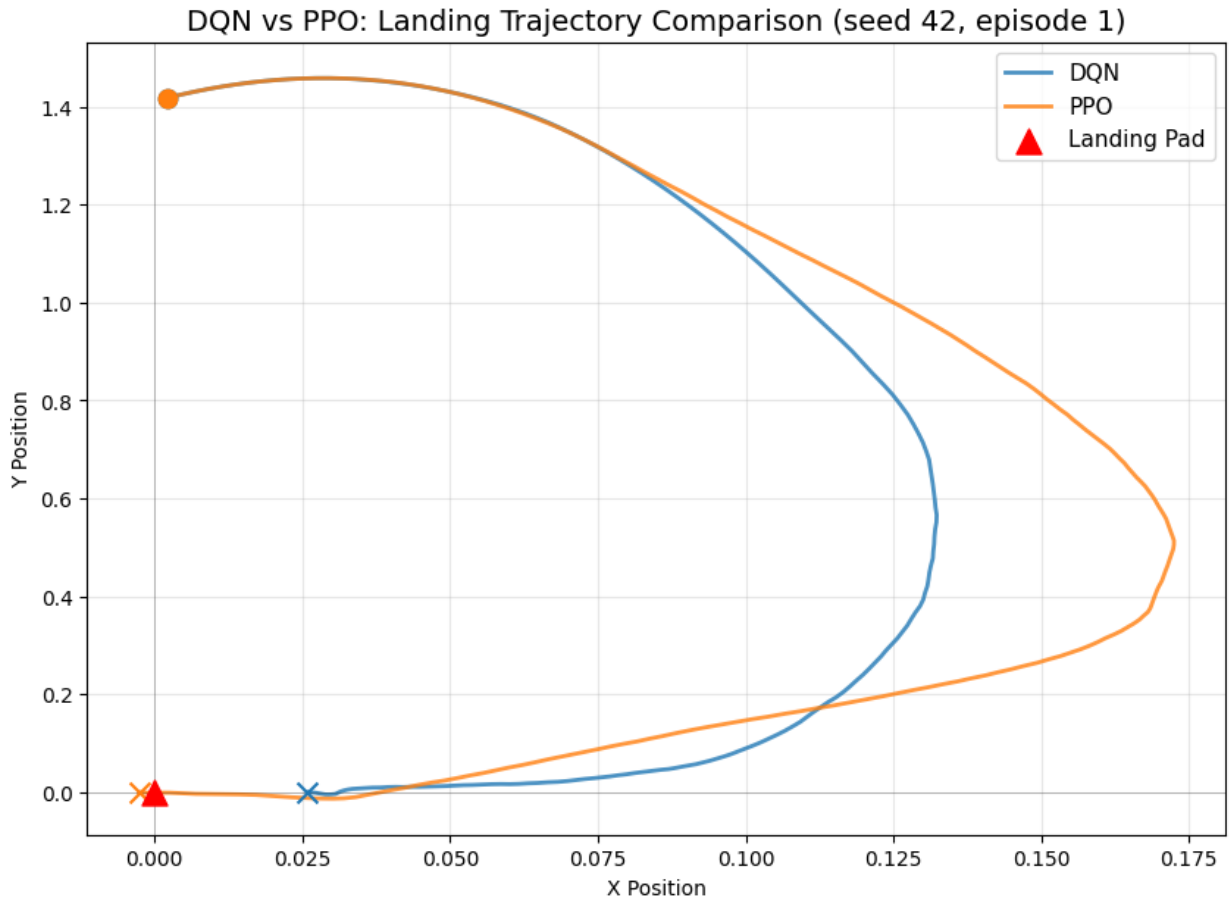



```
# Trajectory Plots: x-y paths of the lander

fig, axes = plt.subplots(1, len(algo_names), figsize=(8 * len(algo_names), 6), sh
arey=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, zo
rder=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 Pa
d')

    ax.set_title(f"{algo_name.upper()} Trajectories (seed {SEED_LIST[0]})", fonts
ize=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 ea
```

```
ch)", 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=8
0, 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]}, episo
de 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 mode
l)...")

        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]), d
evice=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

Property	Value
	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_le
g]")
env_tmp.close()
Environment: LunarLander-v3
Observation space: Box([ -2.5        -2.5       -10.        -10.         -6.28318
55 -10.
  -0.         -0.       ], [ 2.5        2.5       10.        10.         6.283185
5 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 availabl
e'}")
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
```