The City College of New York

CSC 59866-E: Senior Project I

AI Agents for Decision Making in the Real World

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu, sbandyopadhyay@gc.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at the City University of New York

March 23, 2026 CSC 59866

Evolutionary Learning, Mean-Field Learning, Self-play for self-improvement and Continual Learning: Communication paradigms for AI Agents to Adapt in Real-Time Decentralized AI: When and How to Scale AI Agents.

Logistics and Motivation

Recall Lecture 15: We explored theoretical scaling limits, Mean-Field approximation, and Evolutionary Learning for decentralized AI.

We have spent 15 lectures mostly on the *mathematics and theory* of Reinforcement Learning, Game Theory, and Control.

Today's Agenda: How do we actually build this?

Today's Agenda

The Modern Agent Stack: What libraries do researchers use?

Google Colab Setup: Cloud GPUs for training.

Coding the Environment (Gymnasium): Translating MDPs to code.

Coding the Brain (PyTorch): Translating Bellman equations to Tensors.

Multi-Agent Coding (PettingZoo): Setting up CTDE architectures.

The AI Agent Tech Stack

Modern Tools for AI Agents

You do not need to code everything from scratch in raw C++!

The Neural Network Engine: PyTorch /JAX. Handles the calculus, backpropagation, and tensor matrix multiplications automatically.

The Environment Standard: Farama Gymnasium (formerly OpenAI Gym). The universal API for defining states, actions, and rewards.

The Multi-Agent Expansion: PettingZoo. The multi-agent equivalent of Gymnasium.

The Distributed Scaler: Ray RLlib. Used in industry to parallelize agents across thousands of cloud CPUs.

Why Google Colab?

Training a Deep Q-Network or PPO agent on a standard MacBook CPU can take days.

Google Colab: Provides free, cloud-based Jupyter Notebooks with direct access to NVIDIA T4 (or better) GPUs.

Reproducibility: You can share the notebook link with your project partners, ensuring everyone has the exact same Python environment and package versions.

Demo 1 Setup: Go to colab.research.google.com, create a new notebook, and run !pip install gymnasium torch numpy in the first cell.

Coding the Environment

The Gymnasium API (Translating the MDP)

In Lecture 4, we defined an environment mathematically as a Markov Decision Process. Here is how that translates to Python:

The State Space (S): env.observation_space (e.g., a Box of pixel values or a discrete grid).

The Action Space (A): env.action_space (e.g., Discrete(4) for Up, Down, Left, Right).

The Transition Function (P) & Reward (R): Handled entirely by env.step(action).

Mathematical Grounding: The step() Function

When you call next_state, reward, terminated, truncated, info = env.step(action), the engine is sampling from the probability distribution:

$$s_{t+1} \sim P(\cdot|s_t, a_t)$$
$$r_t = R(s_t, a_t, s_{t+1})$$

terminated vs truncated:

- terminated: The agent mathematically reached a terminal state (e.g., won the game, crashed the car).
- truncated: The agent ran out of time (an artificial time limit we impose to stop infinite loops).

Colab Demo 1: Interaction Loop

```
import gymnasium as gym
env = gym.make("CartPole-v1")
state, info = env.reset()

for t in range(1000):
    # Agent chooses a random action (for now)
    action = env.action_space.sample()

    # The Environment reacts
    next_state, reward, terminated, truncated, info =
env.step(action)

    if terminated or truncated:
        state, info = env.reset() # Episode over, restart
    else:
        state = next_state
```

https://colab.research.google.com/drive/15_tYPA2j6fAnGOBaYtwa3VEzpZdgCA9C?usp=sharing

Coding the “Brain”: PyTorch / JAX

Building the Deep Q-Network

We learned that a DQN takes a State vector as input and outputs a Q-Value for every possible Action.

In PyTorch, we define this by inheriting from torch.nn.Module:

```
import torch
import torch.nn as nn

class DQN(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(DQN, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(state_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, action_dim) # Outputs Q-values
        )
    def forward(self, x):
        return self.network(x)
```

Bellman Loss

Recall the Bellman Target from Lecture 8:

$$y_i = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_{\text{target}})$$

And the Mean Squared Error (MSE) Loss function:

$$L(\theta) = \mathbb{E}\left[(y_i - Q(s_t, a_t; \theta))^2\right]$$

PyTorch automatically calculates the gradient $\nabla_\theta L(\theta)$using a computational graph.

Colab Demo 2: Loss in PyTorch

Here is how we translate the exact math from into PyTorch Tensors:

```
# 1. Get current Q-values for the actions we actually took
current_q_values = policy_net(state_batch).gather(1, action_batch)

# 2. Calculate the Max Q-value for the next states (using Target Net)
with torch.no_grad(): # No gradients needed for target!
    max_next_q_values = target_net(next_state_batch).max(1)[0]

# 3. Calculate Bellman Target: r + gamma * max Q
expected_q_values = reward_batch + (gamma * max_next_q_values)

# 4. Compute MSE Loss and Backpropagate
loss = nn.MSELoss()(current_q_values, expected_q_values.unsqueeze(1))
optimizer.zero_grad()
loss.backward()  # Calculates dL/dTheta
optimizer.step() # Updates the weights
```

Scaling to Multi-Agent Coding

Scaling to Multi-Agent (PettingZoo)

When you transition to MARL for your Senior Projects, Gymnasium is no longer enough.

We use **PettingZoo** (also maintained by the Farama Foundation).

AEC vs Parallel: PettingZoo supports both turn-based games (Agent Environment Cycle API, like Chess) and simultaneous-action games (Parallel API, like Autonomous Driving).

The Key Code Difference: Instead of `env.step(action)`, you pass a dictionary mapping every agent's ID to its action: `env.step({"agent_1": action_1, "agent_2": action_2})`.

CTDE in Tensors

Recall **CTDE (Centralized Training, Decentralized Execution)** from MADDPG.

How is this coded? Through Tensor Concatenation.

Actor Input: shape = (batch_size, local_obs_dim)

Critic Input: During training, we gather the observations and actions of *all* agents and concatenate them along the feature dimension using torch.cat.

```
# Centralized Critic sees everything during training!
global_state = torch.cat([obs_agent_1, obs_agent_2], dim=1)
global_actions = torch.cat([act_agent_1, act_agent_2], dim=1)
critic_q_value = central_critic(global_state, global_actions)
```

In a Parallel MARL environment, we iterate through active agents using dictionaries:

```
from pettingzoo.sisl import multiwalker_v9
env = multiwalker_v9.parallel_env()
observations, infos = env.reset()

while env.agents:
    # Get actions for all agents from their respective policies
    actions = {agent: policy(observations[agent]) for agent in
env.agents}

    # Step the environment forward
    observations, rewards, terminations, truncations, infos =
env.step(actions)

    # Store experience in replay buffer for training later...
```

Conclusion

Debugging AI Agents

Traditional code fails with a syntax error. RL code fails *silently*.

Reward Hacking: The agent finds a bug in your env.step() math and exploits it to get infinite reward without solving the task.

Loss vs. Reward: In supervised learning, loss going down is good. In RL, your Bellman Loss might go *up* while your agent gets better, because it is discovering larger rewards and the Q-values are suddenly misaligned. **Always track your Total Episodic Reward**, not just your network loss.

Seed Everything: In Colab, always set torch.manual_seed(42) and env.reset(seed=42). If you don't, you will never be able to reproduce a successful training run.

Summary

The Stack: Google Colab (Compute) + PyTorch (Brain) + Gymnasium/PettingZoo (Environment).

The Translation: Math equations like Bellman Targets translate directly into tensor operations.

The Architecture: Multi-agent structures like CTDE require managing dictionaries of states and concatenating global tensors.

Questions?