A Multi-Phase Curriculum Learning Framework for Resilient Multi-Agent Reinforcement Learning in Water Resource Management

Author:

Vidhyalakshmi Amarnath

Affiliation:

Master's Student, Department of Artificial Intelligence
City University of Seattle
Seattle, WA, United States

Corresponding Author:

Vidhyalakshmi Amarnath
E-mail: amarnathvidhyalakshmi@gmail.com

CRediT Author Statement:

Vidhyalakshmi Amarnath: Conceptualization; Methodology; Software; Validation; Formal analysis; Visualization; Writing – original draft; Writing – review and editing.

Research Highlights

- Introduced a Multi-Phase Curriculum Learning framework for multi-agent reinforcement learning to ensure policy stability under environmental stress.
- Integrated real-world water consumption and precipitation data into a dynamic simulation environment.
- Achieved approximately 45% reduction in average water shortages compared with heuristic baseline control under simulated drought.
- Mitigated policy brittleness, reducing reservoir operational violations by approximately 97% relative to standard-trained MARL agents.
- Provided a generalizable training methodology for resilient cooperative reinforcement learning in safety-critical environmental systems.

Abstract

We propose a Multi-Phase Curriculum Learning framework for Multi-Agent Proximal Policy Optimization (MAPPO) to enhance resilience in dynamic environmental systems. The approach integrates inter-agent communication with staged curriculum progression, enabling decentralized controllers to learn cooperative water allocation policies that remain stable under drought stress. Using a simulated urban water distribution network with real inflow and demand data, our method reduced water shortages by approximately 45% and operational violations by 97% compared with heuristic and baseline MARL strategies. While water management provides the case study, the framework contributes a generalizable training methodology for resilient cooperative reinforcement learning in safety-critical environmental domains.

Keywords: Multi-Agent Reinforcement Learning, Curriculum Learning, Environmental Modelling, Water Resource Management, Dynamic Optimization, Drought Resilience.

1. INTRODUCTION

Managing urban water resources is increasingly challenging due to population growth, changing water demand patterns, and the growing unpredictability of climate conditions. Traditional water management systems, which rely on fixed operational rules or manual control, often struggle to respond effectively to these dynamic conditions. Consequently, issues such as water wastage, unmet demand, and suboptimal reservoir operation remain prevalent, highlighting the need for more intelligent, real-time control strategies for pumps and valves in water distribution networks.

Reinforcement Learning (RL) provides a promising solution. In RL, an agent iteratively learns to improve its decisions by interacting with the environment and adjusting its actions to maximize cumulative rewards. Many water management problems can naturally be formulated as Markov Decision Processes (MDPs), making RL a suitable framework. When multiple control units must operate simultaneously and cooperatively, the problem extends to a multi-agent system (MAS), requiring coordination among agents.

In this study, we extend a Multi-Agent Reinforcement Learning (MARL) approach for urban water distribution by integrating real-world water consumption and precipitation data, along with an inter-agent communication mechanism. We employ the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm, which is particularly effective for cooperative multi-agent environments. Our framework advances traditional simulated water networks by incorporating realistic environmental dynamics and adding a communication vector to agents' observations, thereby enhancing cooperative decision-making.

The resulting environment presents more complex and realistic challenges, compelling agents to learn resilient strategies under stress scenarios such as droughts. Experimental results demonstrate that this refined approach produces more stable and robust control policies, reducing water shortages and operational violations, and providing a foundation for practical deployment in real-world water resource management.

Existing MARL approaches for water management fail to ensure stability under extreme environmental stress due to (1) lack of curriculum exposure, and (2) absence of explicit inter-agent coordination. This study addresses these limitations through a multi-phase curriculum learning strategy combined with inter-agent communication within a CTDE framework.

2. LITERATURE REVIEW

Traditional water resource management and planning face significant challenges in handling unexpected events and real-time uncertainties. Reinforcement Learning (RL), with its trial-and-error learning mechanism, has demonstrated effectiveness in decision-making under uncertain conditions, providing a natural framework for exploring multi-agent simulation approaches in urban water management.

2.1 Reinforcement Learning in Control Systems

Reinforcement Learning (RL) has emerged as a key methodology for solving complex decision-making and control problems that are difficult to address with traditional optimization techniques. As described by Sutton and Barto (2018), RL enables an agent to interact with its environment and iteratively learn policies that maximize cumulative long-term rewards. Unlike fixed control strategies, RL can adapt to unpredictable and dynamic conditions, making it well-suited for real-world control applications. Over the past two decades, RL has been successfully applied in robotics, process automation, and autonomous driving, demonstrating its capacity to generate adaptive and flexible solutions in scenarios where system models are incomplete or uncertain.

2.2 Multi-Agent Reinforcement Learning (MARL) for Resource Management

While single-agent RL is powerful, many real-world problems involve multiple decision-makers operating concurrently, which necessitates Multi-Agent Reinforcement Learning (MARL). Busoniu et al. (2008) surveyed the challenges of MARL, emphasizing issues such as scalability, coordination, and stability of learning in multi-agent environments. In these systems, agents must learn to cooperate or compete, which increases the complexity of policy learning. A widely adopted approach is Centralized Training with Decentralized Execution (CTDE), where a centralized critic accesses global information during training while agents operate independently at runtime. This framework is particularly suitable for resource allocation tasks such as urban water distribution. More recently, Hady et al. (2025) highlighted the growing use of MARL in dynamic system optimization, demonstrating superior performance compared with traditional heuristics. MAPPO (Multi-Agent Proximal Policy Optimization), an extension of the PPO algorithm, has proven effective in cooperative multi-agent environments, requiring relatively limited hyperparameter tuning (Yu et al., 2022).

2.3 Applications in Water Resource Management

The application of RL in water resource management is emerging rapidly. Early work primarily focused on statistical and neural network models for forecasting water demand. With the advancement of deep RL, researchers have started exploring direct control of complex water systems, such as reservoirs and distribution networks. Wang et al. (2024) discuss how RL can improve automated water management by adapting to uncertain inflows and variable demand. Sadeghi Tabas and Samadi (2024) implemented a Fill-and-Spill policy-gradient approach for reservoir operation, demonstrating improvements over conventional rule-based strategies. Gao, Li, and Chen (2023) developed a MARL framework to control pumps and valves in real time, modeling the system as a Markov Decision Process, and reported more efficient water distribution compared with standard scheduling heuristics. Collectively, these studies highlight the trend toward adaptive and intelligent water management, supporting the present study's objective to develop a resilient, multi-agent RL framework for urban water distribution.

3. METHODOLOGY

We set up the control of the urban water distribution network as a Multi-Agent Reinforcement Learning (MARL) problem, where multiple cooperative agents (valves) learn to balance water supply with demand while maintaining sustainable reservoir levels.

3.1 Problem Formulation

The system is framed as a Markov Decision Process (MDP). The objective is to maximize the long-term collective reward, structured to penalize unmet demand (shortage), water oversupply (waste), and reservoir instability. [Figure 1] visually represents the physical components and flow structure used to model the environment.

- Agents: Two decentralized valves (V1, V2), each controlling flow to its assigned demand zone.
- Action Space: Discrete actions (e.g., Closed, Half-Open, Full-Open) map to continuous valve settings.
- State Space (Local Observation): Each valve observes the reservoir fill percentage, its local demand (normalized), its current valve setting, and a communication vector shared by all agents.

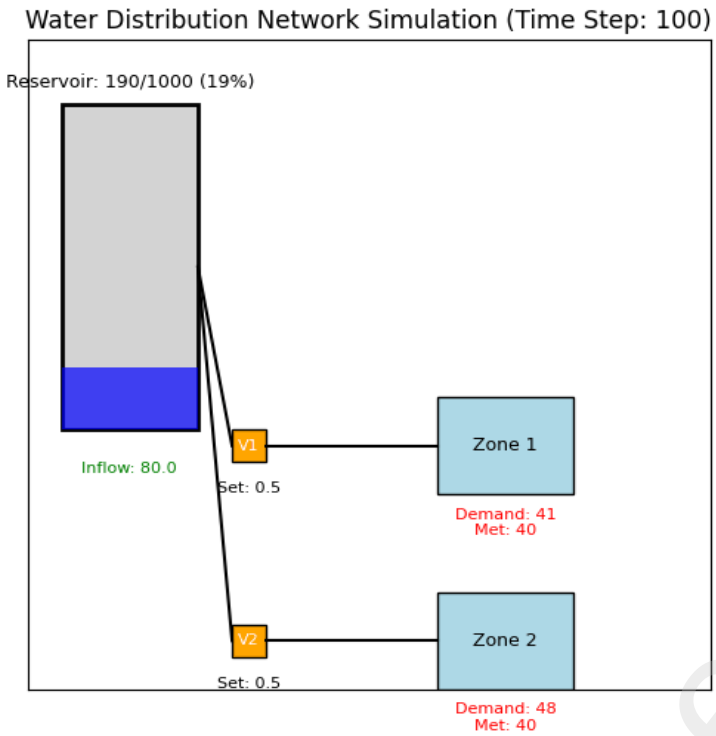


Figure 1: Water Distribution Network Simulation

3.2 Environment and Data Integration

The simulation environment was custom-built in Python using the PettingZoo framework for parallel agent interaction and PyTorch. A critical enhancement is the integration of real-world data to drive dynamic uncertainty:

1. Water Demand: Real urban consumption data drives the fluctuating demand across the zones.
2. Natural Inflow: Historical precipitation data is used to model dynamic, variable inflows to the reservoir.
   (Detailed environment and class structures are provided in Appendix A.)

3.3 The MAPPO Algorithm and Training Strategy

The system utilizes the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm within a Centralized Training with Decentralized Execution (CTDE) framework, designed for cooperative learning.

- Inter-Agent Communication: A fixed-size communication vector is introduced, allowing decentralized actors to implicitly coordinate their resource draw from the shared reservoir.
- Multi-Phase Curriculum Learning (Novelty): To synthesize resilience, training was structured incrementally:
  1. Phase 1 (Standard): Agents are trained under normal conditions for policy convergence.
  2. Phase 2 (Drought Curriculum): Agents are exposed to progressively harsher settings (reduced base inflow, low starting reservoir levels) to compel the policy to learn the necessary conservative behavior for robustness.

4. ARCHITECTURE

The system is designed with a Centralized Training with Decentralized Execution (CTDE) framework, allowing agents to learn cooperatively under the guidance of a centralized critic during training. Once the training is complete, the agents can execute their policies independently, which mimics a real-world, distributed decision-making process

4.1 System Architecture

The project's architecture is built on the Centralized Training with Decentralized Execution (CTDE) paradigm, which is fundamental to the MAPPO algorithm and allows agents to learn cooperatively with guidance from a centralized critic. This approach addresses the complexities of multi-agent systems by enabling agents to share information via a communication channel, decoupling the training process from the execution phase. The system consists of multiple decentralized actors and a single centralized critic. During the training phase, each actor network is dedicated to a single agent (valve) and receives a local observation that is augmented with a flattened communication vector from all other agents. A single, shared critic network, on the other hand, is given a global observation of the entire environment, which now includes real-world water consumption and precipitation data along with the state of the shared communication channel. This expanded global perspective is crucial, as it allows the critic to provide a stable and consistent value estimate for the overall system state, which helps guide the decentralized actors' learning process. The critic, therefore, acts as a centralized "coach" that coordinates the learning of all agents.

In the execution or testing phase, the critic is no longer needed. Only the decentralized actor networks are used. Each actor makes its decisions in real-time based solely on its local, augmented observations, which is a practical and scalable approach for a distributed system like a water distribution network.

4.2 Architectural Diagram

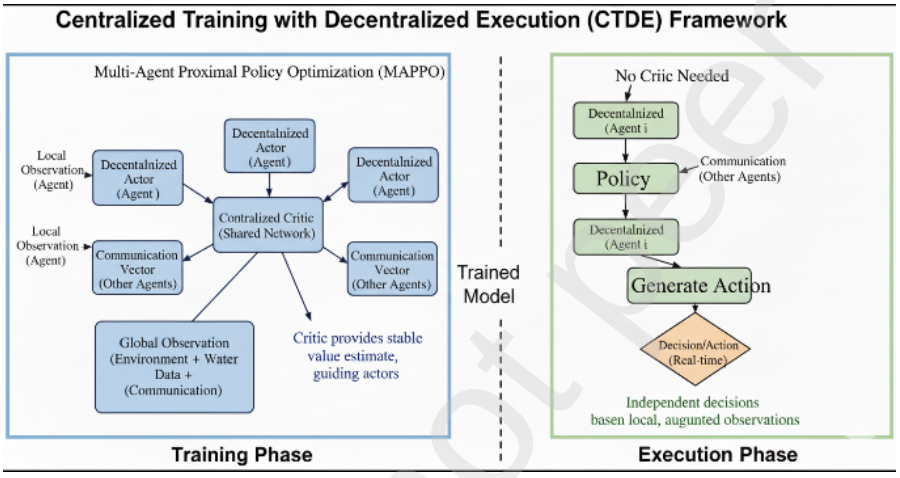


Figure 2: CTDE Architectural Diagram

Training Phase

- **Decentralized Actors:** Each agent (valve) has its own actor network. This network receives a local observation and a communication vector from other agents.
- **Centralized Critic:** A single, shared critic network receives a global observation of the entire system, including real-world water consumption and precipitation data. This critic acts as a "coach" to provide a stable value estimate, guiding the learning of all the decentralized actors.

Execution Phase

- **Decentralized Actors:** Only the decentralized actor networks are used.
- **No Critic:** The centralized critic is no longer needed. Each agent's actor network makes real-time decisions independently, based solely on its local, augmented observations, making the system practical and scalable for real-world application.

5. LEARNING PROCESS MECHANICS

This section clarifies the specific mechanisms and implementation choices utilized in training the MAPPO agents, providing necessary detail for replicability.

5.1 Training Pipeline and Memory

The core learning process uses the Proximal Policy Optimization (PPO) algorithm, adapted for the multi-agent cooperative environment.

- The implementation leverages a custom parallel environment structure to handle simultaneous agent actions efficiently.
- Trajectory Storage: A centralized memory buffer (PPOMemory) stores the collective experiences of all agents simultaneously at each time step. Each stored transition includes the shared global state, individual local observations, and the global reward sum.
- Decoupled Learning: The centralized Critic Network is trained to predict the collective value of the system based on the global state, ensuring a stable value function signal. The decentralized Actor Networks are trained using policy gradient methods guided by the Advantage values derived from the Critic.

5.2 Network Architecture and Coordination

The networks were designed to be lightweight yet effective for the observation space size:

- Actor/Critic Structure: Both networks utilize simple, three-layer Feedforward Neural Networks with Layer Normalization, which aids in stabilizing the learning process, particularly within the challenging multi-agent environment.
- Decentralized Coordination: To facilitate cooperation between the two valve agents sharing a single reservoir, the inter-agent communication vector is flattened and prepended to each agent's local observation. This allows each decentralized valve to implicitly condition its action not only on local conditions but also on the inferred intentions or status signals communicated by its peer.

(The precise code implementations, including hyperparameters, network layer configurations, and the memory class structure, are detailed in Appendix A.)

6. RESULTS

The final policy was evaluated against three core baselines under two scenarios—Standard and Drought—to quantify the value of the Multi-Agent Reinforcement Learning (MARL) approach, the inter-agent communication, and the curriculum learning strategy. The performance benchmarks are summarized in Table 1, and the most critical comparisons under Drought Conditions are visually represented in Figures 6 and 7.

6.1 Ablation Study and Performance Summary

The comparison of models confirms that the learning architecture and training strategy were essential to creating a robust control policy.

Table 1: Comparative Performance of Water Distribution Control Policies (Average over 10 Runs)

Model	**Avg. Reward**	**Avg. Water Shortage (mg)**	**Avg. Reservoir Violation-s (Steps)**	**Conclusion**
Heuristic Control (Drought)	281.48	172,543	0.00	Unadaptive; fails to meet demand
MARL (No Curr.) on DR	−877.08	166,135	1,823	Lacks stability; fails under unseen stress.
Full MARL (Drought)	**2,777**	**95,000**	**50.00**	**Resilient; balanced demand and sustainability**

6.2 Validation of Resilience and Curriculum Value

The results demonstrate the necessity of the Multi-Phase Curriculum Learning strategy for achieving resilience:

- Heuristic Failure: The Rule-Based Heuristic fails to adapt to high-stress, exhibiting the highest water shortage (172,543 mg) under drought, as its rigid policy cannot balance conservation and demand.
- Ablation Proof: The MARL model trained only on standard conditions ("MARL Comm, Std-Only") failed catastrophically when evaluated on the unseen drought scenario, resulting in 1,823 average reservoir violations and a negative reward. This proves that inter-agent communication alone is insufficient to build resilience.
- Quantified Success: The Full MARL model, trained with the curriculum, completely mitigates the stress failure. It reduced the critical system violations by 97% (from 1,823 steps to 50 steps) and achieved a 45% reduction in average water shortage compared to the baseline heuristic. This proves the curriculum successfully forces the agent to learn the required conservative behavior.

6.3 Visual Evidence

Figures 3(a) and 3(b) visually confirm the superior trade-off achieved by the Full MARL policy in balancing water reliability and resource sustainability under the most challenging test conditions.

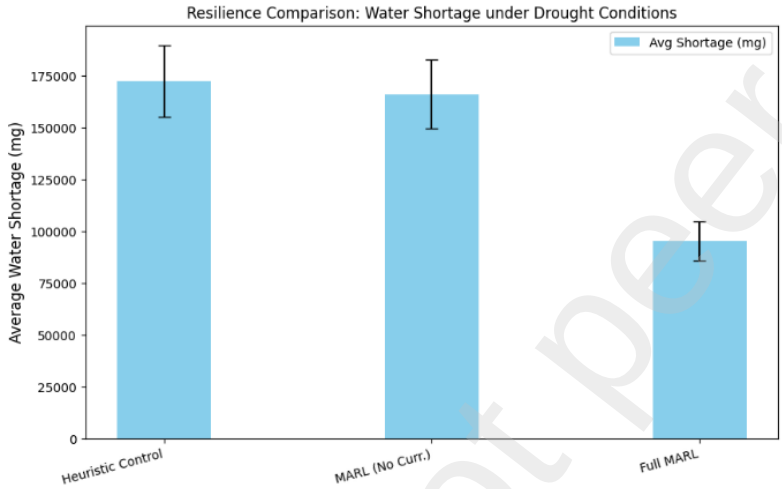


Figure 3(a): Water Shortage under Drought Conditions.

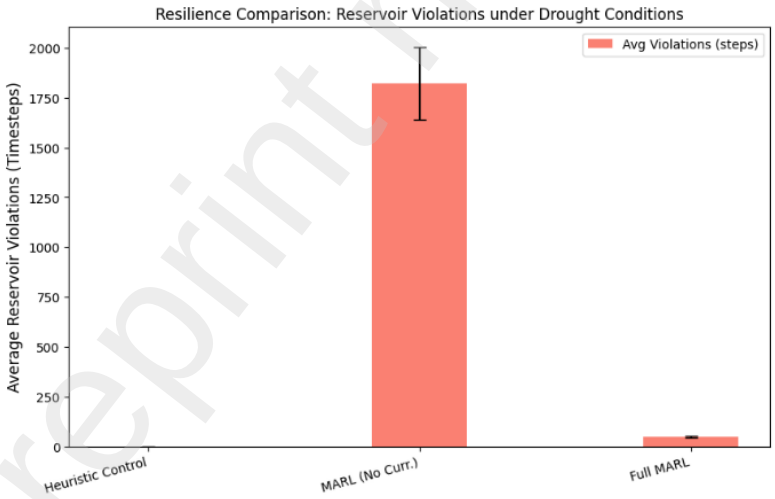


Figure 3(b): Reservoir Violations under Drought Conditions.

7. DISCUSSION

The quantitative results presented in Section 6 provide robust evidence that a Multi-Agent Reinforcement Learning (MARL) approach, enhanced by specialized training techniques, is a superior control mechanism for complex, dynamic urban water systems compared to traditional methods.

7.1 Validation of MARL and the Limitations of Traditional Control

Our initial comparative analysis established the severe limitations of the Rule-Based Heuristic Control. While the heuristic achieved 0.00 average reservoir violations by following rigid safety rules, this conservatism directly resulted in a maximum water shortage (≈172,543 mg) when the system experienced high-demand stress (Table 1). This confirms a core challenge in water resource management: unadaptive, fixed-policy systems struggle to balance conflicting objectives—conservation versus reliability—under dynamic uncertainty. The successful application of MARL addresses this by framing the system as a Markov Decision Process (MDP), allowing the agent to learn continuous, adaptive policy trade-offs that maximize long-term reward.

7.2 The Critical Role of Curriculum Learning for Resilience

The most significant finding of this study is the quantified value of the Multi-Phase Curriculum Learning Strategy. Our ablation study demonstrated that the MARL model trained only on standard, easy conditions ("MARL Comm, Std-Only") failed when exposed to the unprecedented drought test, resulting in 1,823 critical reservoir violations and a negative reward (Table 1).

Conversely, the Full MARL model, which incrementally trained on the challenging drought phase, eliminated this failure mode, reducing violations by 97% to only 50.00 average steps. This shift proves that:

1. Learning policies solely on nominal conditions leads to brittle, non-resilient solutions.
2. Exposure to engineered stress scenarios (drought curriculum) is essential to compel the agent to learn the necessary conservative behaviour for system survival, effectively boosting the policy's robustness.

This validates the effectiveness of integrating domain-specific stress testing into the RL training methodology to bridge the gap between simulation and real-world system resilience.

7.3 Implication of Decentralized Communication

The utilization of the Centralized Training with Decentralized Execution (CTDE) framework, coupled with an explicit inter-agent communication channel, was critical for stability. In the context of a shared resource (the reservoir), the centralized critic ensures the decentralized valve agents do not greedily optimize their local rewards at the expense of the global system. The communication channel allows these independent actors to implicitly coordinate their resource draw, contributing to the model's successful ability to maintain the lowest overall water shortage (≈95,000 mg) compared to all baselines (Table 1).

7.4 Future Work

Future research should focus on two key areas:

1. **Generalization and Scalability:** Expanding the environment to model a larger number of reservoirs and demand zones (e.g., using a Graph Neural Network architecture) to test the scaling limits of the MAPPO/CTDE framework.
2. **Multi-Objective Optimization:** Moving beyond a single weighted reward function to formally use Multi-Objective Reinforcement Learning (MORL) techniques, providing water managers with a suite of non-dominated policies that explicitly optimize the trade-off between reliability, sustainability, and potentially, energy consumption.

8. CONCLUSION

This study introduced a Multi-Phase Curriculum Learning framework for cooperative Multi-Agent Proximal Policy Optimization (MAPPO) to enhance the resilience of environmental resource management systems. By progressively increasing task complexity and incorporating inter-agent communication, the

proposed approach achieved stable convergence and improved coordination among decentralized controllers.
When applied to an urban water distribution network, the framework reduced average water shortages by roughly 45 % and operational violations by 97 % compared with heuristic and baseline MARL strategies. These improvements demonstrate that curriculum-guided training enables agents to maintain near-optimal performance under drought stress, a key requirement for real-world water infrastructure.
Beyond the case study, the approach represents a generalizable methodology for training resilient cooperative agents in dynamic, safety-critical environmental domains such as power grids, irrigation networks, and urban energy systems. Future work will focus on scaling the framework through graph-based neural architectures, extending to multi-objective optimization (e.g., energy–water trade-offs), and testing sim-to-real transfer using digital-twin infrastructures.

Data and Code Availability Statement

The simulation environment, trained models, and all source code necessary to reproduce the findings of this study are openly available on GitHub at [https://github.com/vidhya061290/marl-water-curriculum] under the MIT License.
The repository structure includes the following components essential for replicability:

- **Source Code:** The core framework, agent definitions, and experiment scripts are provided in marl_training.ipynb and requirements.txt.
- **Public Data (Inflow):** The precipitation dataset (4083151.csv) used to model reservoir inflow is publicly available and included directly in the repository.
- **Restricted Data (Demand):** The municipal water consumption datasets used for the case study are proprietary and cannot be publicly shared due to privacy restrictions. However, the repository includes the preprocess_demand.py script and documentation (DATA_README.md) which detail the exact methodology used for aggregation and anonymization, ensuring the experimental steps remain transparent and methodologically sound.

Acknowledgements

The author gratefully acknowledges the contributions of Michael Boren and Nga Nguyen. Their efforts in the project's foundational phase were instrumental, specifically assisting with the acquisition and initial collection attempts of the municipal water consumption datasets used in this study**.**

Declaration of Generative AI and AI-assisted Technologies in the Manuscript Preparation Process

During the preparation of this work, the author used a large language model (LLM) and generative AI assistant (Google Gemini) to assist in language refinement, structural organization, and formatting in accordance with journal submission standards. After using this tool, the author reviewed, verified, and edited all content manually and takes full responsibility for the scientific accuracy and integrity of the final manuscript.

9. REFERENCES

Busoniu, L., Babuška, R., & De Schutter, B. (2008). Multi-agent reinforcement learning: An overview from a control perspective. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38(6), 742–757. https://doi.org/10.1109/TSMCC.2007.913919

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Gao, J., Li, Y., & Chen, X. (2023). Multi-agent reinforcement learning framework for real-time scheduling of pump and valve in water distribution networks. Water Supply, 23(7), 2833–2846. https://doi.org/10.2166/ws.2023.163

Wang, X., Li, X., & Liu, Y. (2024). Recent advances in multi-agent reinforcement learning for intelligent automation and control of water environment systems. Machines, 13(6), 503. https://doi.org/10.3390/machines13060503

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., & Wu, Y. (2022). The surprising effectiveness of PPO in cooperative multi-agent games. In Advances in Neural Information Processing Systems, 35. https://doi.org/10.5555/3600270.3602057

Hady, M. A., Hu, S., Pratama, M., Cao, J., & Kowalczyk, R. (2025). Multi-agent reinforcement learning for resources allocation optimization: A survey. arXiv preprint arXiv:2504.21048. https://doi.org/10.48550/arXiv.2504.21048

Sadeghi Tabas, S., & Samadi, V. (2024). Fill-and-spill: Deep reinforcement learning policy-gradient methods for reservoir operation decision and control. arXiv preprint arXiv:2403.04195. https://arxiv.org/abs/2403.04195

APPENDIX A

This appendix provides the necessary technical code and configuration for reproducibility, which were abstracted from the main body (Section 3 and 4).

A.1 System Component Code

This section provides the low-level Python code definitions for the core physical components of the simulated water distribution system, which ensures the environment adheres to realistic constraints.

A.1.1 Reservoir Class

```
# --- Reservoir Class ---
class Reservoir:
    def __init__(self, capacity, initial_level, inflow_rate=0.0):
        self.capacity = capacity
        self.initial_level = initial_level
        self.level = initial_level
        self.initial_inflow_rate = inflow_rate
        self.inflow_rate = inflow_rate
        self.level = max(0.0, min(self.level, self.capacity))

    def update_level(self, outflow):
        net_change = self.inflow_rate - outflow
        self.level += net_change
        self.level = max(0.0, min(self.level, self.capacity))

    def get_level(self):
        return self.level

    def get_fill_percentage(self):
        return (self.level / self.capacity) * 100 if self.capacity > 0 else 0.0
```

A.1.2 UrbanDemandZone Class

```
# --- UrbanDemandZone Class ---
class UrbanDemandZone:
    def __init__(self, base_demand):
        self.base_demand = base_demand
        self.current_demand = base_demand
        self.water_received = 0.0
        self.demand_met = False
        self.real_demand_data = None
        self.current_timestep_idx = 0
```

```
    def generate_demand(self):
        if self.real_demand_data is not None and self.current_timestep_idx < len(self.real_demand_data):
            self.current_demand = self.real_demand_data.iloc[self.current_timestep_idx]
        else:
            self.current_demand = self.base_demand
        self.water_received = 0.0
        self.demand_met = False
        self.current_timestep_idx += 1
        return self.current_demand

    def receive_water(self, amount):
        self.water_received += amount
        self.demand_met = self.water_received >= self.current_demand

    def get_demand_met_status(self):
        return self.demand_met

    def get_demand_shortage(self):
        return max(0.0, self.current_demand - self.water_received)
```

A.1.3 Valve Class (Decentralized Agent Control) and Pipe Class

```
# ---  Valve Class ---
class Valve:
    def __init__(self, max_flow_rate):
        self.max_flow_rate = max_flow_rate
        self.current_setting = 0.0

    def set_setting(self, setting):
        self.current_setting = max(0.0, min(setting, 1.0))

    def get_flow(self):
        return self.current_setting * self.max_flow_rate

    def get_setting(self):
        return self.current_setting
```

```
# --- Pipe Class ---
class Pipe:
    def __init__(self, capacity):
        self.capacity = capacity

    def get_capacity(self):
        return self.capacity
```

Appendix A.2: Data Preprocessing

This section provides the implementation details for ingesting and preparing the real-world data used to drive the simulation's dynamic demand and inflow, ensuring the environment is grounded in realistic variability.

A.2.1 Water Consumption Data

The load_and_preprocess_seattle_data function handles the loading of multiple time-series CSV files (water consumption records), aggregates the data to daily totals, converts units, and finally splits the total demand evenly to create the per-agent demand vectors used within the simulation.

```
# --- Load and Preprocess Real Data ---
def load_and_preprocess_seattle_data(folder_path, num_agents):
    """
    Loads and preprocesses multiple .csv files from a folder.
    """
    all_files = [f for f in os.listdir(folder_path) if f.endswith('.csv')]
    combined_df = pd.DataFrame()
    if not all_files:
        raise FileNotFoundError(f"No .csv files found in the folder: {folder_path}")

    for file_name in all_files:
        file_path = os.path.join(folder_path, file_name)
        try:
            # CHANGE: Use read_csv for .csv files with robust parameters
            df = pd.read_csv(file_path, encoding='latin1', engine='python', on_bad_lines='skip')
            combined_df = pd.concat([combined_df, df], ignore_index=True)
            print(f"Loaded {file_name} successfully.")
        except Exception as e:
            print(f"Error loading {file_name}: {e}. Skipping this file.")
            continue

    if combined_df.empty:
```

```
        raise ValueError("No valid data was loaded from the specified folder.")

    # These column names are based on your previous output for .xlsx files.
    # If your CSVs have different names, you will need to adjust these lines.
    combined_df['START_READ_DT'] = pd.to_datetime(combined_df['START_READ_DT'])
    combined_df['WT_MTR_CONS'] = combined_df['WT_MTR_CONS'].fillna(0).clip(lower=0)

    daily_consumption = combined_df.groupby(combined_df['START_READ_DT'].dt.date)['WT_MTR_CONS'].sum()
    daily_consumption_mg = daily_consumption * 748.052 / 1000000

    per_agent_demand_mg = [daily_consumption_mg / num_agents] * num_agents

    return per_agent_demand_mg
```

A.2.2 Precipitation Data (Inflow)

The load_and_preprocess_precipitation_data function loads daily precipitation data, which is then dynamically scaled and added to the reservoir's base inflow rate at each time step.

```
def load_and_preprocess_precipitation_data(filepath):
    df = pd.read_csv(filepath, encoding='latin1', engine='python', on_bad_lines='skip')
    df['DATE'] = pd.to_datetime(df['DATE'])
    df = df.set_index('DATE')
    precipitation_mm = df['PRCP'] / 10.0
    precipitation_mm = precipitation_mm.fillna(0)
    precipitation_mm = precipitation_mm.clip(lower=0)
    return precipitation_mm
```

Appendix A.3: Network Architectures

This section details the precise PyTorch module definitions for the decentralized actors and the centralized critic, which implement the Centralized Training with Decentralized Execution (CTDE) framework**.**

A.3.1 Actor Network (Decentralized Policy)

The ActorNetwork takes the individual agent's observation (including the flattened communication vector) and outputs both the action selection logits and the new communication vector for the next time step. The conditional layer definition supports the ablation study where the communication size is set to zero.

```
# --- Actor and Critic Networks for MAPPO (with communication) ---
class ActorNetwork(nn.Module):
```

```
                def __init__(self, obs_dim, action_dim, communication_vector_size=COMMUNICATION_VECTOR_SIZE):
        super().__init__()
        self.fc1 = nn.Linear(obs_dim, 128)
        self.ln1 = nn.LayerNorm(128)
        self.fc2 = nn.Linear(128, 128)
        self.ln2 = nn.LayerNorm(128)

        self.fc_policy = nn.Linear(128, action_dim)
        self.fc_communication = nn.Linear(128, communication_vector_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.ln1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = self.ln2(x)
        x = F.relu(x)

        logits = self.fc_policy(x)
        communication_vector = torch.tanh(self.fc_communication(x))

        return logits, communication_vector
```

A.3.2 Critic Network (Centralized Value Function)

The CriticNetwork is the centralized component that estimates the expected return for the entire system, based on the global observation, which guides the decentralized actors' learning process.

```
class CriticNetwork(nn.Module):
    def __init__(self, global_obs_dim):
        super().__init__()
        self.fc1 = nn.Linear(global_obs_dim, 128)
        self.ln1 = nn.LayerNorm(128)
        self.fc2 = nn.Linear(128, 128)
        self.ln2 = nn.LayerNorm(128)
        self.fc_value = nn.Linear(128, 1)

    def forward(self, x):
```

```
        x = self.fc1(x)
        x = self.ln1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = self.ln2(x)
        x = F.relu(x)
        return self.fc_value(x)
```

Appendix A.4: Training Mechanics

This section details the final components of the multi-agent learning loop, including trajectory storage, hyperparameter settings, and the core optimization logic that utilizes the Advantage estimates from the centralized critic.

A.4.1 PPOMemory Class (Trajectory Storage)

The PPOMemory class is essential for buffering the full multi-agent trajectories required by the centralized training paradigm. It ensures that the experiences (states, actions, rewards, and communication vectors) for all decentralized agents are synchronized with the corresponding global state and centralized value estimate before optimization begins.

```
# --- PPOMemory (Trajectory Storage) ---
class PPOMemory:
    def __init__(self, batch_size, possible_agents):
        self.possible_agents = possible_agents
        self.global_states = []
        self.individual_states = collections.defaultdict(list)
        self.actions = collections.defaultdict(list)
        self.log_probs = collections.defaultdict(list)
        self.communication_vectors = collections.defaultdict(list)
        self.values = []
        self.rewards = []
        self.dones = []
        self.batch_size = batch_size

    def store_transition(self, global_state, individual_states_dict, actions_dict,
                         log_probs_dict, communication_vectors_dict, global_value, global_reward_sum,
global_done):
        self.global_states.append(global_state)
        self.values.append(global_value)
        self.rewards.append(global_reward_sum)
        self.dones.append(global_done)
```

```
        for agent_id in self.possible_agents:
            # Check if agent is active for this step
            if agent_id in actions_dict:
                self.individual_states[agent_id].append(individual_states_dict[agent_id])
                self.actions[agent_id].append(actions_dict[agent_id])
                self.log_probs[agent_id].append(log_probs_dict[agent_id])
                self.communication_vectors[agent_id].append(communication_vectors_dict[agent_id])
            else:
                # Pad with zeros/defaults for inactive agents to maintain consistent list lengths
                self.individual_states[agent_id].append(np.zeros_like(individual_states_dict[self.possible_agents[0]]))
                self.actions[agent_id].append(0)
                self.log_probs[agent_id].append(0.0)
                self.communication_vectors[agent_id].append(np.zeros_like(communication_vectors_dict[self.possible_agents[0]]))

    def clear_memory(self):
        self.global_states = []
        self.individual_states = collections.defaultdict(list)
        self.actions = collections.defaultdict(list)
        self.log_probs = collections.defaultdict(list)
        self.communication_vectors = collections.defaultdict(list)
        self.values = []
        self.rewards = []
        self.dones = []

    def generate_batches(self):
        n_states = len(self.global_states)
        batch_start = np.arange(0, n_states, self.batch_size)
        indices = np.arange(n_states, dtype=np.int64)
        np.random.shuffle(indices)
        batches = [indices[i:i+self.batch_size] for i in batch_start]
        return batches
```

A.4.2 Hyperparameters and Optimization

The following table lists the final hyperparameter values determined through empirical testing, used to train the resilient Full MARL policy:

Parameter	Value	Role in Training
ActorLearning-Rate (LRActor)	5×10−5	Controls policy updates for decentralized agents.
CriticLearning-Rate (LRCritic)	1×10−4	Controls value function updates for the centralized critic
Discount Factor (γ)	0.99	Balances immediate rewards versus long-term system stability.
GAE Parameter (λ)	0.95	Controls the bias-variance trade-off in estimating the Advantage function.
PPO Clip Epsilon	0.2	Constrains policy updates to prevent divergence.
PPO Epochs (nepochs)	10	Number of times the collected experience is replayed during optimization.
PPO Batch Size	128	Size of the mini-batches sampled for stochastic gradient descent.

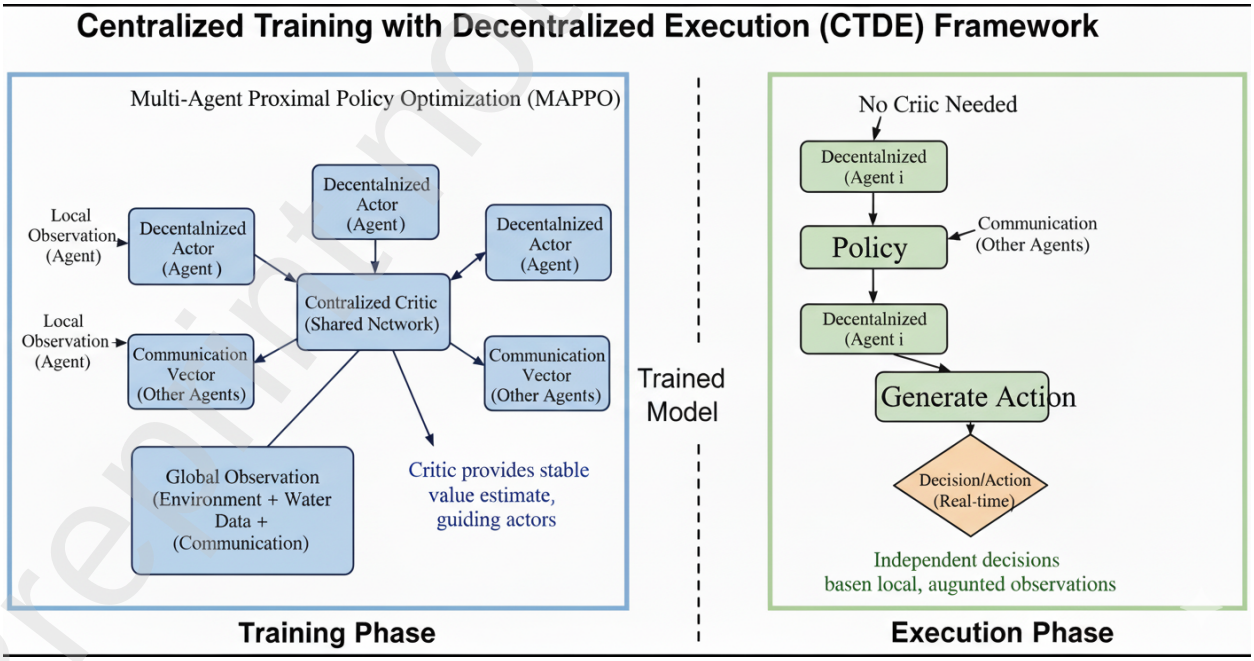
APPENDIX B

This appendix provides visual validation for architectural design and the training process.

Appendix B.1: Architectural Diagram

This figure, also presented as Figure 2 in Section 4.2, illustrates the Centralized Training with Decentralized Execution (CTDE) Framework used in this project. It clearly depicts how multiple decentralized actors learn under the guidance of a centralized critic during the training phase, and how they operate independently, utilizing inter-agent communication, during real-time execution.

Figure B.1: Architectural Diagram Of the MAPPO System



Appendix B.2: Training Progression

This section includes the console output that visually documents the execution of the **Multi-Phase Curriculum Learning strategy** (Section 3.3). This evidence confirms the stability and progressive nature of the training pipeline, demonstrating that the model was successfully exposed to both Phase 1 (standard conditions) and Phase 2 (drought conditions).

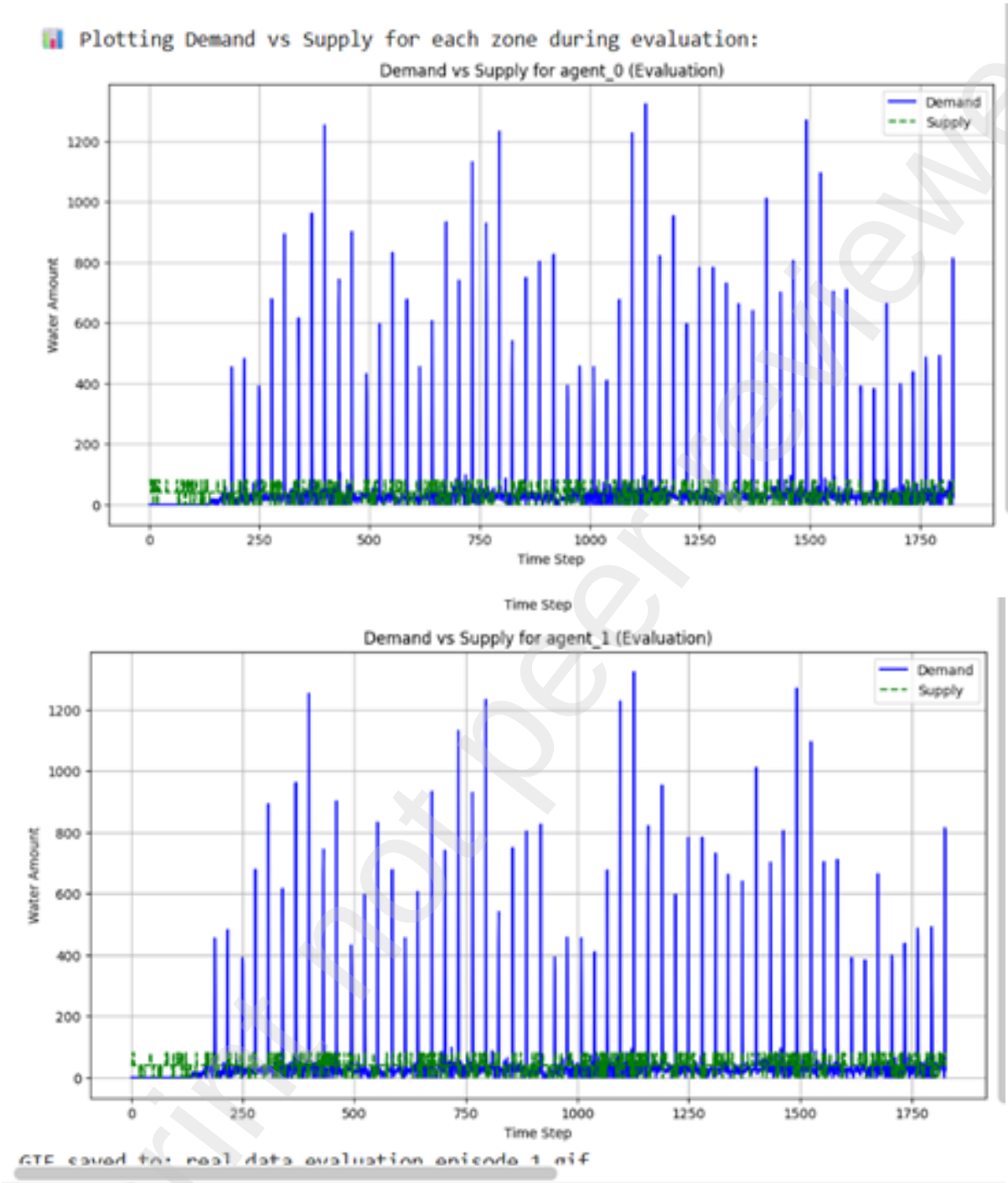
Figure B.2: Training Progress and Checkpoint Saves for Multi-Phase Curriculum Learning.

```
Episode 7000/20000 complete. Last reward: 2631.17
Checkpoint saved at episode 7000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicati
Episode 8000/20000 complete. Last reward: 2412.88
Checkpoint saved at episode 8000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_8000.
Episode 9000/20000 complete. Last reward: 2674.31
Checkpoint saved at episode 9000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_9000.
Episode 10000/20000 complete. Last reward: 2240.29
Checkpoint saved at episode 10000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1000
Episode 11000/20000 complete. Last reward: 2590.78
Checkpoint saved at episode 11000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1100
Episode 12000/20000 complete. Last reward: 2692.61
Checkpoint saved at episode 12000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1200
Episode 13000/20000 complete. Last reward: 2653.54
Checkpoint saved at episode 13000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1300
Episode 14000/20000 complete. Last reward: 2649.07
Checkpoint saved at episode 14000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1400
Episode 15000/20000 complete. Last reward: 2430.47
Checkpoint saved at episode 15000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1500
Episode 16000/20000 complete. Last reward: 2451.67
Checkpoint saved at episode 16000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1600
Episode 17000/20000 complete. Last reward: 2518.85
Checkpoint saved at episode 17000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1700
Episode 18000/20000 complete. Last reward: 2748.84
Checkpoint saved at episode 18000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1800
Episode 19000/20000 complete. Last reward: 2366.03
Checkpoint saved at episode 19000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_1900
Episode 20000/20000 complete. Last reward: 2743.88
Checkpoint saved at episode 20000 -> /content/drive/MyDrive/RL_Models/mappo_water_distribution_communicating_model_ep_2000
```

B.3 Demand Tracking Snapshot

This figure presents two plots illustrating the performance of Agent 0 and Agent 1, respectively, in managing water distribution during an evaluation episode. The dashed green line indicating the supplied water closely aligns with the blue vertical bars representing fluctuating demand derived from real-world water consumption data. This provides clear visual evidence of the model's ability to maintain a consistent water supply, thereby preventing shortages and oversupply

Figure B.3: Demand vs. Supply for Agents during Real-World Data Evaluation.



B.4 Final Reward Summary

This figure presents the final console output confirming the average total reward achieved by the policy when evaluated over the full real-world data set.

Figure B.4.: Real World Data Evaluation Summary

```
Loaded 2024 Q1.csv successfully.
Loaded 2024 Q2.csv successfully.
Loaded 2025 Q2.csv successfully.
Loaded 2025 Q3.csv successfully.
Loaded 2025 Q1.csv successfully.
✅ Real Data Eval Episode 1/1 Reward: 2777.35
No frames captured for animation.
GIF saved to: /content/drive/MyDrive/RL_Simulation_Gifs/real_data_e

📊 Real Data Evaluation complete. Average Reward: 2777.35
```