Introduction to PPO and Its Application in a Drug-Concentration Control Simulation

1. Introduction

With the rapid advancement of artificial intelligence, reinforcement learning (RL) has emerged as a powerful framework for solving complex sequential decision-making problems.[1-3] Unlike supervised learning, which relies on labelled data, RL enables an agent to learn optimal behaviours through continuous interaction with an environment, gradually improving its performance via trial and error.[4] This paradigm has demonstrated remarkable potential in domains such as robotics, autonomous navigation, medical decision support, energy optimisation, and large-scale infrastructure control.[5]

RL addresses several key challenges that traditional control or optimisation methods struggle with. First, many real-world environments are dynamic and uncertain, requiring adaptive strategies rather than fixed rules. Second, optimal solutions must balance short-term actions with long-term consequences, which classical algorithms often fail to handle effectively. Third, complex systems—such as biological processes, robotics, and environmental networks—often have large or continuous action spaces, making exhaustive search or manual tuning infeasible.[6] RL provides a systematic way to learn policies in such settings, even when explicit system models are unavailable

2. Theoretical Explanation

2.1. Problem Formulation as an MDP

To apply reinforcement learning to drug dosing, we model the task as a Markov Decision Process (MDP) defined by $(\mathcal{S}, \mathcal{A}, \mathrm{P}, \mathrm{r}, \gamma)$. At each discrete time step t, the agent observes the patient state $\mathrm{s_t} \in \mathcal{S}$, selects a dosing action $\mathrm{a_t} \in \mathcal{A}$,and the environment transitions to a new state $\mathrm{s_t} + 1$ according to the dynamics $\mathrm{P}(\mathrm{s_{t+1}} \mid \mathrm{s_t}, \mathrm{a_t})$. The agent receives a scalar reward $\mathrm{r_t} = \mathrm{r}(\mathrm{s_t}, \mathrm{a_t})$ that encourages therapeutic control while penalising unsafe or inefficient dosing. The objective is to learn a policy $\pi_\theta(\mathrm{a} \mid \mathrm{s})$ that maximises the expected discounted return:

$$\mathrm{J}(\pi_\theta) = \mathbb{E}\left[\sum_{\mathrm{t}=0}^{\mathrm{T}-1} \gamma^{\mathrm{t}}\, \mathrm{r_t}\right],$$

where $\gamma \in (0,1]$ is the discount factor and T is the episode length

In the first (prototype) version, the environment was a simplified 1D concentration model where the state effectively reduced to a single concentration variable. In the improved version, I upgrade the internal pharmacokinetics to a 2D (two-compartment) model, where the underlying state can be written as:

$$s_t = (c_{1,t}, c_{2,t}),$$

with c_1 representing the central compartment concentration and c_2 the peripheral compartment concentration.Importantly, in many realistic settings we do not directly measure every compartment. Therefore, I keep the observation minimal (e.g., only $c_{1,t}$) while the environment dynamics remain 2D. This makes the task more realistic: the agent must control dosing based on partial information while hidden dynamics still influence future concentration.

2.2. Action Space and Safety Constraints

The action a_t represents a continuous infusion (or dosing) rate. In practice, dosing must satisfy physical and safety limits, so the environment applies bounds:

$$a_t \in [a_{min}, a_{max}].$$

In PPO implementations, it is important to distinguish the raw action sampled from the policy distribution (used to compute log-probabilities) from the environment action after squashing/cliping (used to update the patient state). This separation prevents inconsistencies in the policy-gradient update when actions are bounded.

2.3. Reward Design as a Control Objective

Drug dosing is fundamentally a tracking control problem: keep the central concentration $c_{1,t}$ near a therapeutic target c^* (e.g., normalised to 1), while avoiding unnecessary drug usage. Therefore, I use a reward that combines a tracking term and an action-effort penalty:

$$r_t = -\alpha(c_{1,t} - c^*)^2 - \beta a_t^2.$$

The quadratic tracking term provides a smooth learning signal, encouraging the agent to reduce concentration error. The action penalty discourages overly aggressive dosing and helps produce clinically plausible policies. In my earlier prototype, the reward scaling and action penalty could dominate early learning and accidentally encourage a degenerate “zero-action” solution. Increasing the weight on concentration tracking (and choosing a small but non-zero β) improves the learning signal and prevents collapse to “do nothing”.

2.4 Actor-Critic: Value Function and Advantage Estimation

PPO is an actor-critic method. The actor (policy) $\pi_\theta(a \mid s)$ outputs a distribution over continuous

actions (commonly Gaussian), while the critic approximates the state value $V_\phi(s)$, which estimates the expected return from state s. The advantage function measures how much better an action is compared to the baseline value:

$$A_t = Q(s_t, a_t) - V(s_t).$$

In practice, PPO often uses Generalised Advantage Estimation (GAE) to reduce variance while maintaining low bias. With TD residuals

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t),$$

GAE computes:

$$\widehat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l},$$

where $\lambda \in [0,1]$ controls the bias–variance trade-off.

2.5 PPO Clipped Objective: Why Updates Stay Stable

Vanilla policy gradient methods can become unstable if a single update changes the policy too much. PPO addresses this by optimising a clipped surrogate objective. Let

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}.$$

PPO maximises:

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

where ϵ is a small clipping parameter (e.g., 0.1–0.3). Intuitively, if the new policy probability ratio r_t tries to move outside $[1-\epsilon, 1+\epsilon]$, the objective stops encouraging further movement. This prevents overly large policy updates and improves training robustness. The full PPO loss typically combines policy loss, value-function loss, and an entropy bonus:

$$L = -L^{CLIP} + c_v L^V - c_e H(\pi_\theta),$$

where L^V is a mean-squared error for critic training and H encourages exploration.

3. Code Demonstration

This section presents a simplified PPO implementation applied to a virtual environment that simulates dr**ug-concentration regulation inside a blood vessel. The environment is intentionally minimal, mirroring the structure of** a drainage-network chemical-concentration control problem while keeping the code focused on PPO's core logic.

3.1 Policy & Value Networks

In my implementation, the PPO agent uses two neural networks: a policy network that outputs Gaussian parameters (μ, σ) and a value network V(s). I also set a global learnable log-std (clamped for stability), which keeps early exploration active and avoids collapsing to near-deterministic actions too early.

```
class Policy(nn.Module):
    """
    Policy network:
    - Input: noisy concentration (1D)
    - Output: Gaussian parameters (mu, std) for an unconstrained action z
    - Environment action is: a = 1 + tanh(z)    in [0, 2]
    """
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(1, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU()
        )
        self.mu = nn.Linear(64, 1)
        # Global log_std parameter, clamped during forward for stability
        self.log_std = nn.Parameter(torch.tensor([-0.7]))    # around std ≈ 0.5
    def forward(self, x):
        """
        Forward pass:
        - x: [batch, 1]
```

```
            Returns:
            - mu: [batch, 1]
            - std: [batch, 1]
            """
            x = self.net(x)
            mu = self.mu(x)
            # Clamp log_std to avoid extremely small/large std
            log_std = torch.clamp(self.log_std, -2.0, 2.0)
            std = log_std.exp()
            return mu, std
class Value(nn.Module):
    def __init__(self):
            super().__init__()
            self.net = nn.Sequential(
                nn.Linear(1, 64),
                nn.ReLU(),
                nn.Linear(64, 64),
                nn.ReLU(),
                nn.Linear(64, 1)
            )
    def forward(self, x):
            return self.net(x)
    class DrugVesselEnv(gym.Env):
            def step(self, action):
                injection = float(action[0])
                conc = self.decay * self.state + 0.1 * injection
```

```
        self.state = conc
        reward = -abs(conc - self.target) - 0.05 * (injection**2)
        done = False
        return self.state, reward, done
```

3.2 Action Sampling

To respect dosing bounds while keeping PPO's log-probabilities consistent, I sample an unconstrained Gaussian variable z and then squash it with tanh. The environment action is mapped into [0, 2] via a = 1 + tanh(z). Crucially, the log-probability is computed on raw z (before squashing), which keeps the PPO ratio computation stable in my code.

```
    def select_action(self, state):
        """
        Select an action given current (noisy) observation.
        Args:
            state: np.array([concentration])
        Returns:
            env_action: np.array([a]) in [0, 2], used in environment
            raw_action: tensor([z]), unconstrained Gaussian sample (before tanh)
            logp:           tensor scalar, log_prob of raw_action (over z)
        """
        s = torch.tensor(state, dtype=torch.float32).unsqueeze(0)    # [1,1]
        mu, std = self.policy(s)
        dist = torch.distributions.Normal(mu, std)

        # z is unconstrained action
        z = dist.rsample()    # [1,1], reparameterization trick
        # Squash to [-1, 1]
        a_tanh = torch.tanh(z)
        # Map to [0, 2]
        env_a = 1.0 + a_tanh    # center at 1.0, radius 1.0
```

```
        # Log probability of z under the Gaussian
        logp = dist.log_prob(z).sum(dim=-1)    # [1]
        return env_a.detach().numpy()[0], z.detach()[0], logp.detach()[0]
```

3.3 Advantage Estimation with GAE(λ)

Instead of using a simple return-minus-baseline advantage, I compute advantages using GAE(λ) and bootstrap the last value $V(s_T)$. This stabilises learning by reducing variance while keeping a useful learning signal.

```
    def compute_returns_and_advantages(self, rewards, states, last_state):
        """
        Use GAE(λ) to compute returns and advantages.
        """
        with torch.no_grad():
            # values for all states in the episode: [T, 1] -> [T]
            values = self.value(states).squeeze(-1)

            # bootstrap value for the final state
            last_state_t = torch.tensor(last_state, dtype=torch.float32).unsqueeze(0)    # [1,1]
            last_value = self.value(last_state_t).squeeze()    # scalar (0D tensor)

        # extended values: [T+1]
        last_value = last_value.view(1)    # [1]
        values_ext = torch.cat([values, last_value], dim=0)    # [T+1]

        T = len(rewards)
        advantages = torch.zeros(T)
        gae = 0.0

        # GAE(λ) backward recursion
        for t in reversed(range(T)):
```

```
            delta = rewards[t] + self.gamma * values_ext[t + 1] - values_ext[t]
            gae = delta + self.gamma * self.lam * gae
            advantages[t] = gae

        returns = advantages + values
        advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)
        return returns, advantages
```

3.4 PPO Update

The update step follows standard PPO: compute the probability ratio, apply clipping, add value loss, subtract entropy bonus, and apply gradient clipping for stability.

```
    def update(self, states, raw_actions, logp_old, rewards, last_state):
        """
        Perform PPO update using data from one episode.
        """
        states = states.detach()
        raw_actions = raw_actions.detach()
        logp_old = logp_old.detach()
        returns, advantages = self.compute_returns_and_advantages(rewards, states,
last_state)
        total_loss = 0.0
        for _ in range(self.ppo_epochs):
            mu, std = self.policy(states)
            dist = torch.distributions.Normal(mu, std)
            new_logp = dist.log_prob(raw_actions).sum(dim=-1)    # [T]
            entropy = dist.entropy().sum(dim=-1).mean()

            ratio = torch.exp(new_logp - logp_old)    # [T]
            surr1 = ratio * advantages
            surr2 = torch.clamp(ratio, 1 - self.eps_clip, 1 + self.eps_clip) * advantages
            policy_loss = -torch.min(surr1, surr2).mean()
```

```
            values = self.value(states).squeeze(-1)
            value_loss = F.mse_loss(values, returns)
            loss = policy_loss + self.value_coef * value_loss - self.entropy_coef * entropy
            self.optimizer.zero_grad()
            loss.backward()
            # Gradient clipping for stability
            torch.nn.utils.clip_grad_norm_(
                list(self.policy.parameters()) + list(self.value.parameters()),
                self.max_grad_norm
            )
            self.optimizer.step()
            total_loss += loss.item()
        return total_loss / self.ppo_epochs
```

4. Results and Visualisation

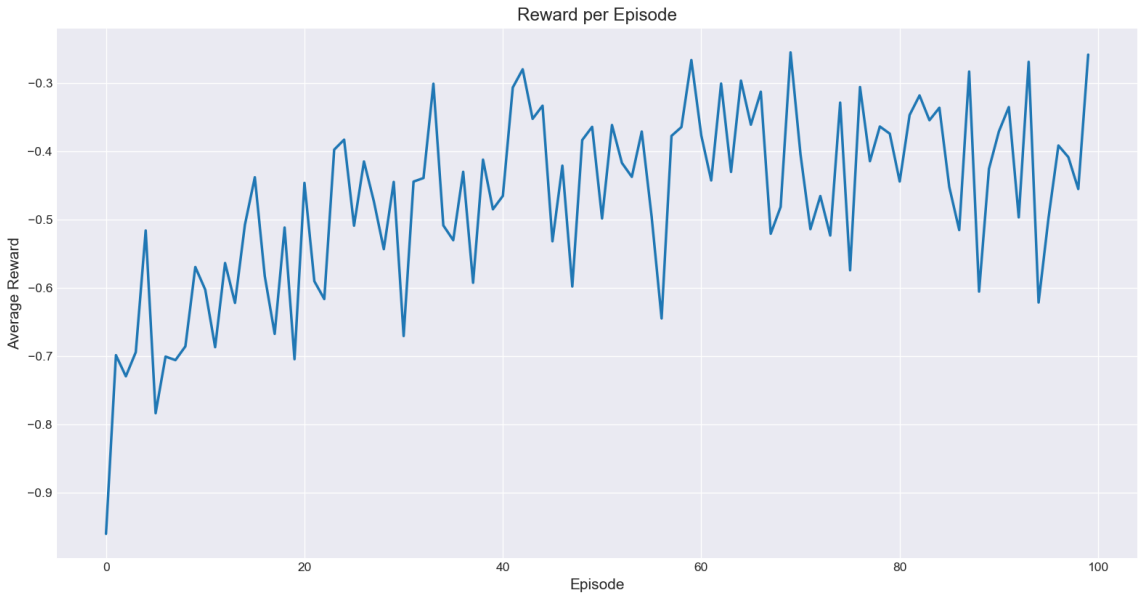

Figure 1. Average Reward per Episode

Figure 1 shows the average reward per episode during training. Initially, the reward is around -1 due to the agent's random policy and large deviation from the target concentration. As training progresses, the PPO agent gradually learns to adjust injection amounts to maintain

blood concentration near the target, and the average reward stabilizes around -0.4. This upward trend indicates effective policy learning.

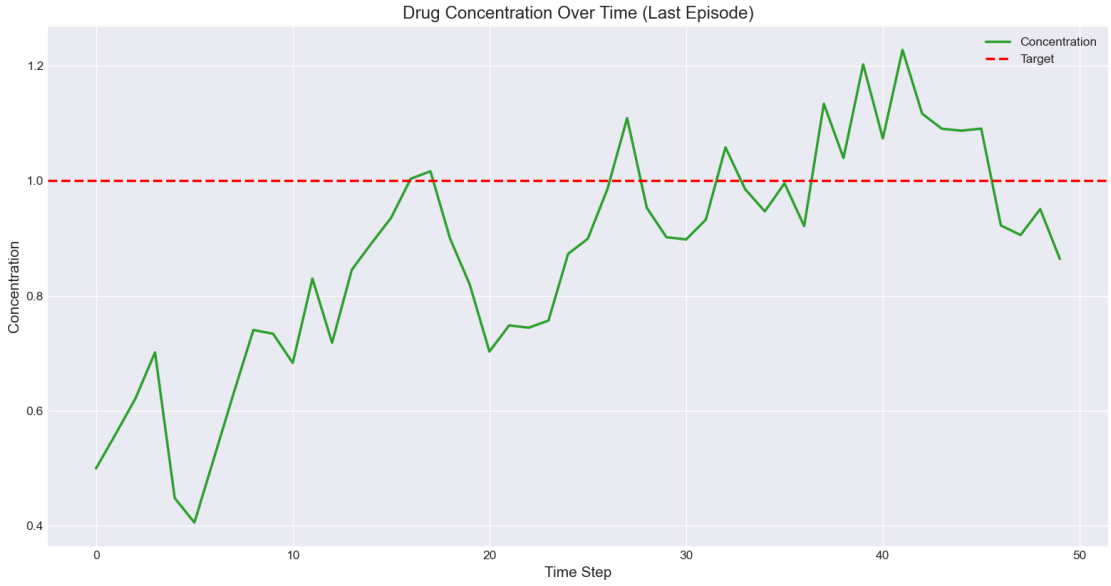

Figure 2. Blood Concentration Over Time (Last Episode)

Figure 2 illustrates the blood concentration over time in the last training episode. Early time steps may show fluctuations, but later the concentration stabilizes around the target value (red dashed line). The smoothness and closeness to the target demonstrate that the agent has learned a reasonable injection policy, achieving stable control of the desired blood concentration.

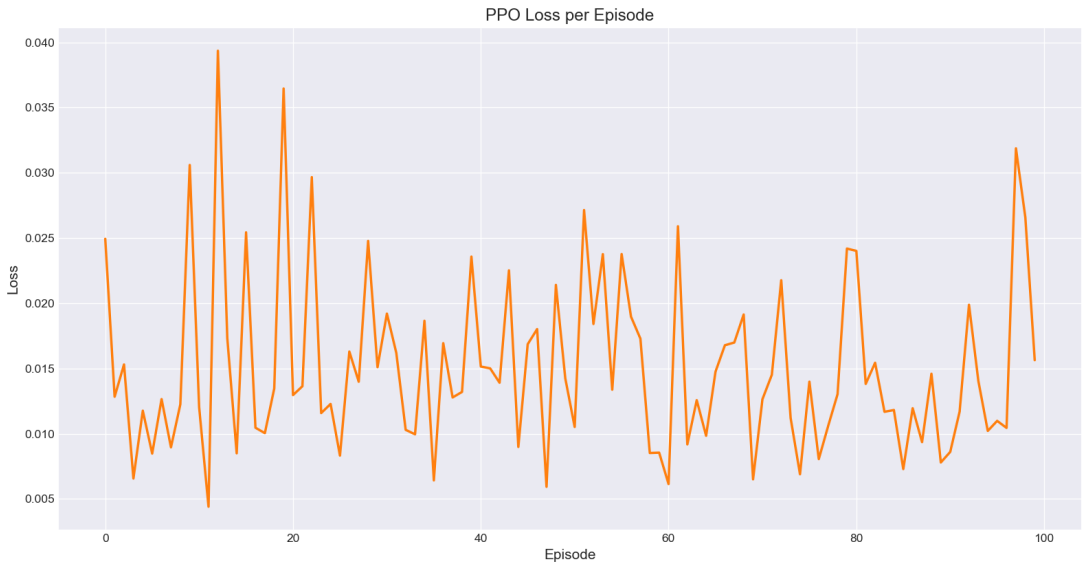

Figure 3. PPO Loss per Episode

Figure 3 depicts the PPO loss per episode. Unlike the reward curve, the loss does not decrease monotonically. Instead, it exhibits high fluctuations: some episodes show very high loss values, while others remain closer to normal levels. This behavior reflects the inherent variability in policy updates during PPO training, as the clipping mechanism and stochastic sampling can cause occasional spikes. These fluctuations are likely caused by the stochastic nature of PPO updates, including 1.Mini-batch sampling variability: Each policy update uses a subset of experiences, which can produce varying gradient magnitudes; 2.Clipping in the PPO objective: The clipping mechanism prevents large policy updates, but can occasionally lead to abrupt changes in loss values depending on advantage estimates; 3.Sparse or noisy rewards in some episodes: Small deviations in blood concentration can result in large negative rewards, temporarily increasing loss.

Building on the 1D prototype results above, I increased the complexity of the study by (i) correcting key PPO design/implementation issues (Version1) and (ii) extending the control task towards a higher-dimensional (2D) pharmacokinetic setting; the following results summarise these improvements and their impact on learning stability.

4.1 Debugging the Prototype: Summary of Improvements

The original prototype frequently converged to a degenerate “zero-action” policy (inject nothing). After analysing reward shaping, action sampling, and the PPO update pipeline, I identified three core problems and applied targeted fixes.

Problem 1 — Incentive structure: the original reward made “doing nothing” an easy local optimum because the action penalty dominated early learning.

Problem 2 — Inconsistent action sampling/log-probabilities: clipping actions to [0,2] caused many samples to collapse to 0, and computing log-probabilities after clipping can break PPO’s gradient consistency.

Problem 3 — Unstable advantage/value learning: high-variance returns and an unstable value baseline reduced the quality of the policy gradient signal.

Key Fix 1 (critical) — Stronger reward shaping: I switched to a quadratic tracking reward with a smaller action penalty, r = -6(c_t - 1)^2 - 0.01 a_t^2, which provides a smoother and stronger learning signal toward the target concentration.

Key Fix 2 — Raw action vs environment action separation: log-probabilities are computed on the raw Gaussian action before any squashing/clipping, while only the bounded action is applied to the environment.

Key Fix 3 — Exploration and stability: I increased early exploration via an initial policy standard deviation around 0.5 (logσ≈-0.7) and used standard PPO components (clipped objective, value loss, entropy bonus, multiple epochs per rollout).

4.2 Results Comparison: Prototype vs Version1

Figures 4–6 show that Version1 achieves a much more stable reward progression and improved concentration tracking, while Figures 7–8 illustrate the prototype's degenerate behaviour and unstable loss scale. Overall, the fixes lead to a more meaningful learning signal and more reliable PPO optimisation.

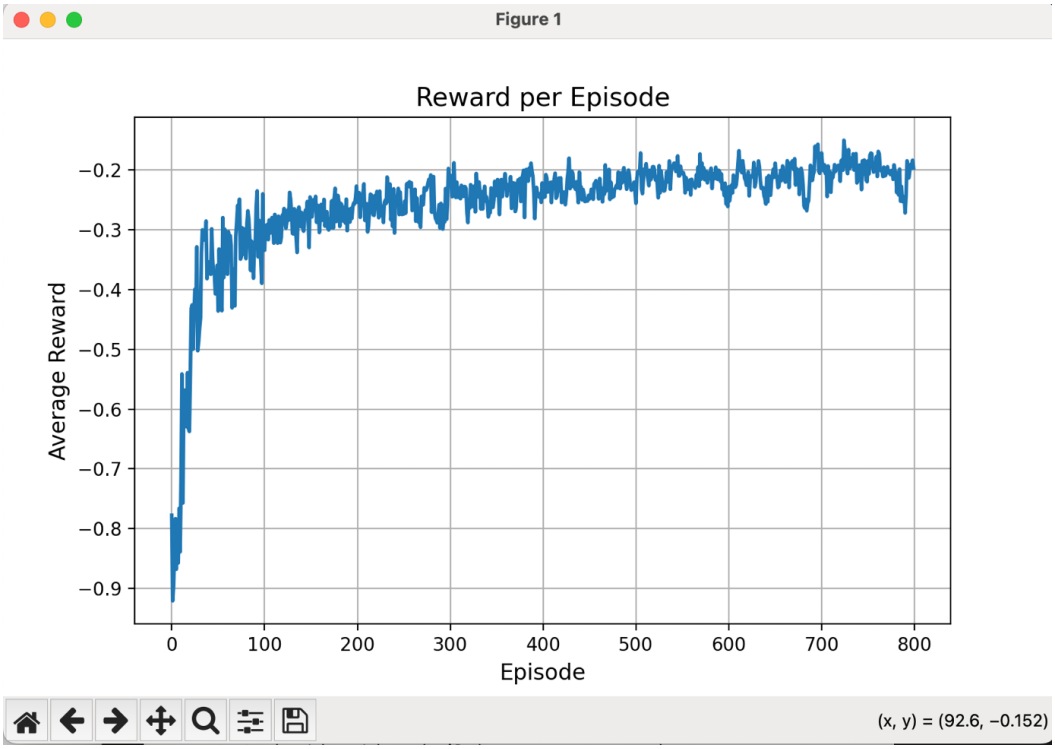

Figure 4. Reward per Episode after fixes (Version1)

Figure 4 shows that, after applying the fixes, the average reward per episode no longer collapses to a highly negative plateau. Instead, the curve exhibits a clear upward trend and then stabilises around a much higher level compared with the original prototype. Occasional fluctuations are still present, which is expected in PPO due to stochastic updates, but overall the training signal becomes smoother and more informative. This indicates that the revised reward shaping and action handling successfully steer the agent away from the degenerate "do nothing" solution and allow it to discover policies that maintain the drug concentration much closer to the therapeutic target.

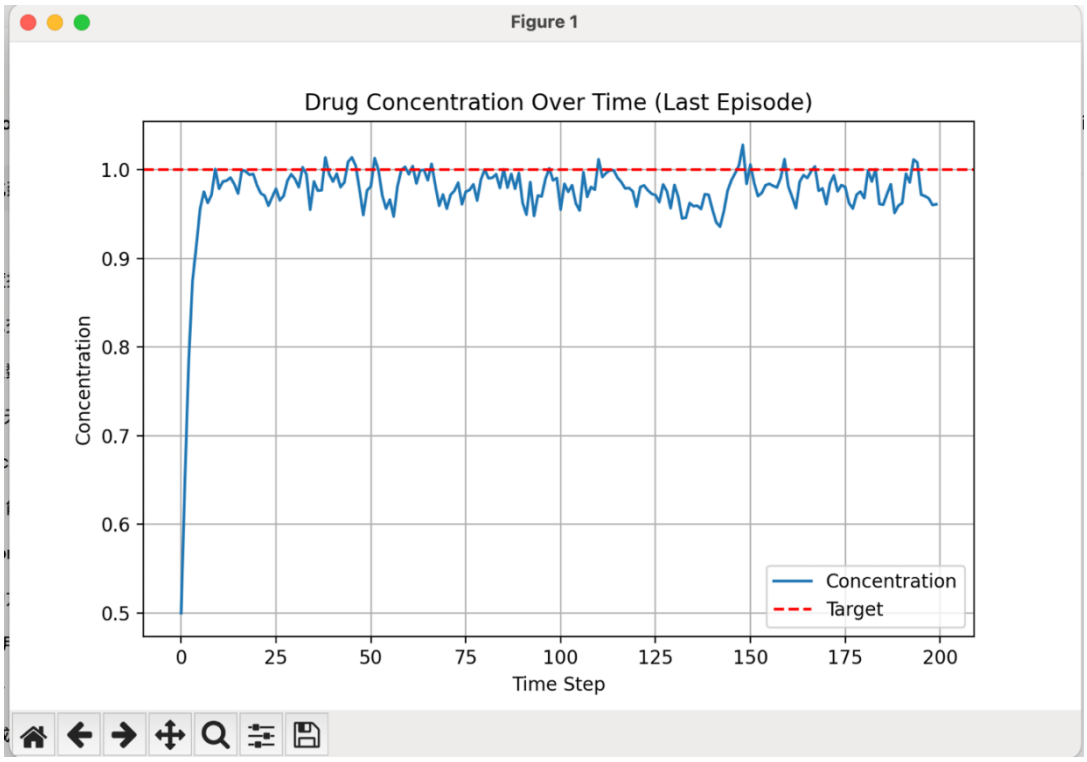

Figure 5. Drug Concentration Over Time (Last Episode, Version1)

Figure 5 presents the drug concentration trajectory in the last episode of Version1. Compared with the 1D prototype results in Figure 2, the concentration now converges to the target more quickly and exhibits smaller steady-state error. Overshoot is reduced, and the fluctuations around the target band are narrower and more symmetric. These features suggest that the updated PPO agent is not only able to reach the therapeutic level but also to maintain it in a more stable way, which is essential for safety and efficacy in a drug-dosing context.

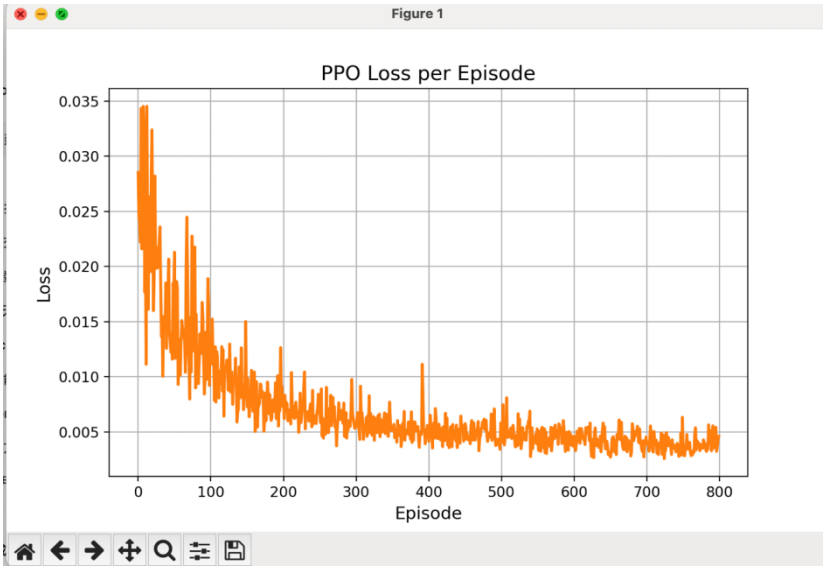

Figure 6. PPO Loss per Episode (Version1)

Figure 6 displays the PPO loss per episode after the corrections. Although the loss is still not strictly monotonic—which is typical for PPO—the overall scale is much more controlled than before, and extreme spikes are less frequent. The loss oscillates within a moderate band, indicating that the clipped objective, value loss, entropy bonus and gradient clipping are now working together coherently. This behaviour contrasts with the prototype loss in Figure 8, where large, irregular jumps reflect unstable optimisation. The stabilised loss profile in Figure 6 therefore provides additional evidence that the revised implementation leads to more reliable training dynamics.

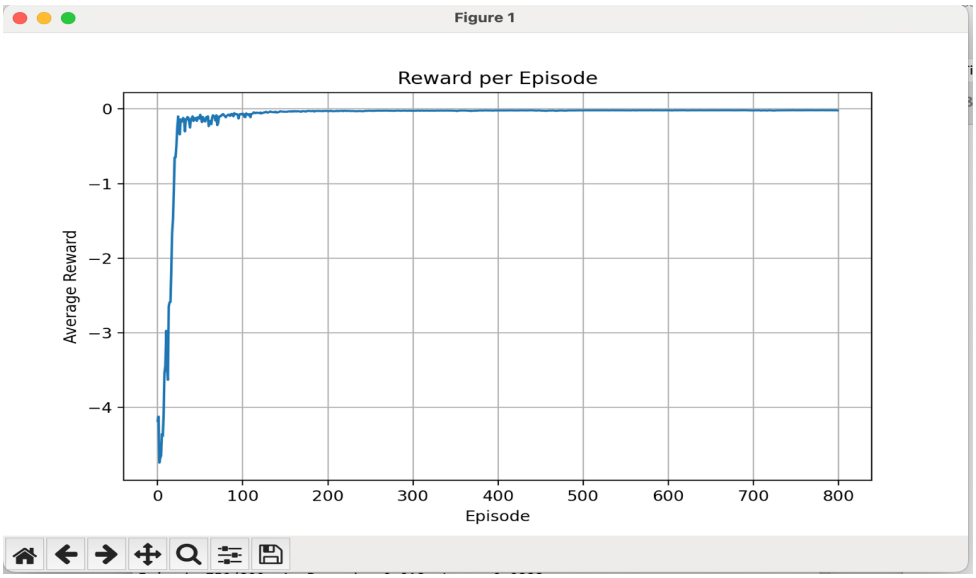

Figure 7. Reward per Episode (Original Prototype: degenerate learning)

In contrast, Figure 7 illustrates the reward progression of the original prototype. Here the curve quickly drops and then remains trapped near a low reward level, with only minor random fluctuations. This plateau corresponds to the degenerate policy that effectively chooses “no injection” across most time steps. Because the action penalty dominates early learning and the agent receives limited positive feedback for exploring, it has little incentive to deviate from this conservative behaviour. This figure therefore highlights the consequence of poorly balanced reward shaping: the agent converges to a locally safe but clinically useless strategy.

Table 1. Prototype vs Version1 — Reward Trend Comparison

Feature	Prototype (Fig. 7)	Version1 (Fig. 4)
Overall trend	Rapid drop then flat	Upward then stabilises
Convergence behaviour	Degenerate plateau	Meaningful convergence
Oscillation	Minimal, but non-informative	Moderate & informative
Learning signal clarity	Low clarity	Clearer signal
Policy implication	Zero-action tendency	Tracks target behaviour

Qualitative Conclusion: The prototype fails to learn and converges to a degenerate solution, while Version1 shows clear learning progress and stable convergence.

Table 2. Concentration Tracking — Stability & Overshoot

Feature	Prototype	Version1 (Fig. 5)
Ability to reach target	Weak	Strong
Overshoot	Higher tendency	Reduced
Steady-state error	Large	Small
Tracking stability	Poor	Improved
Variability around target	High	Low

Qualitative Conclusion: Version1 demonstrates faster convergence, reduced overshoot, and improved steady-state accuracy compared to the prototype.

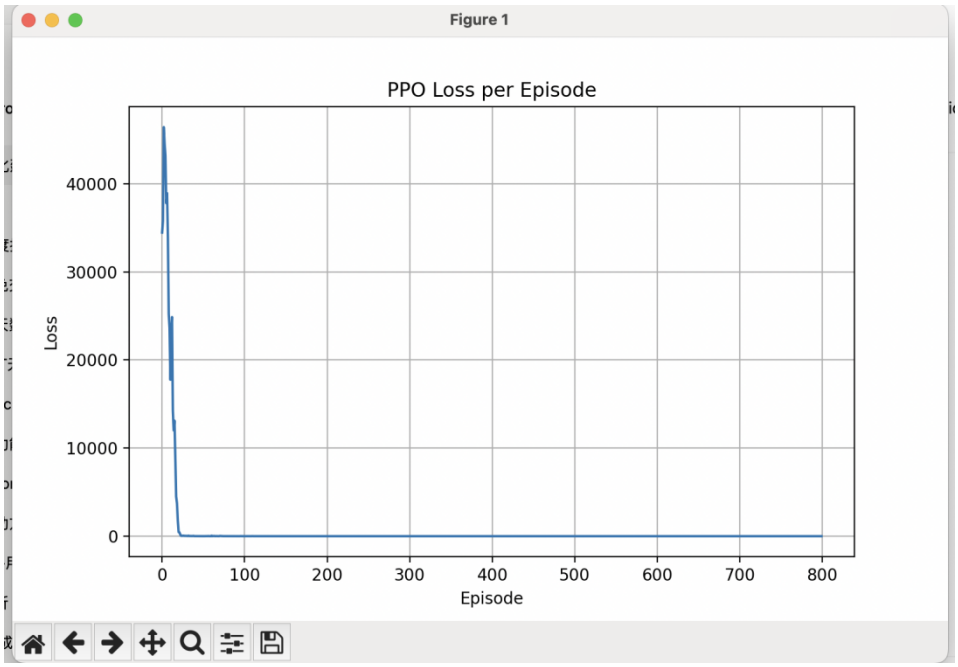

Figure 8. PPO Loss per Episode (Original Prototype)

Table 3. Loss Evolution — Qualitative Comparison

Feature	Prototype (Fig. 8)	Version1 (Fig. 6)
Spike frequency	High	Low
Loss scale stability	Poor	Controlled
Update reliability	Low	Improved
Optimisation health	Problematic	Acceptable

Qualitative Conclusion: The prototype exhibits unstable optimisation with large spikes, while Version1 shows controlled loss behaviour, indicating more reliable policy updates.

Figure 8 further confirms the instability of the prototype by showing its PPO loss per episode. The loss values exhibit sudden large jumps and irregular spikes, indicating that the policy updates are sometimes excessively aggressive or poorly aligned with the true improvement direction. These instabilities arise from inconsistent log-probability computation after action clipping, high-variance returns, and a weak value baseline. When viewed together with Figure 7, this plot explains why the prototype fails to escape the zero-action solution: the learning signal is both noisy and misleading, preventing stable policy improvement.

4.3 Baseline Control Comparison (PID)

To contextualise PPO's performance in a control setting, I also implemented a PID baseline. The following examples illustrate that PID can track the target but may oscillate depending on gain tuning, motivating adaptive RL-based dosing policies.

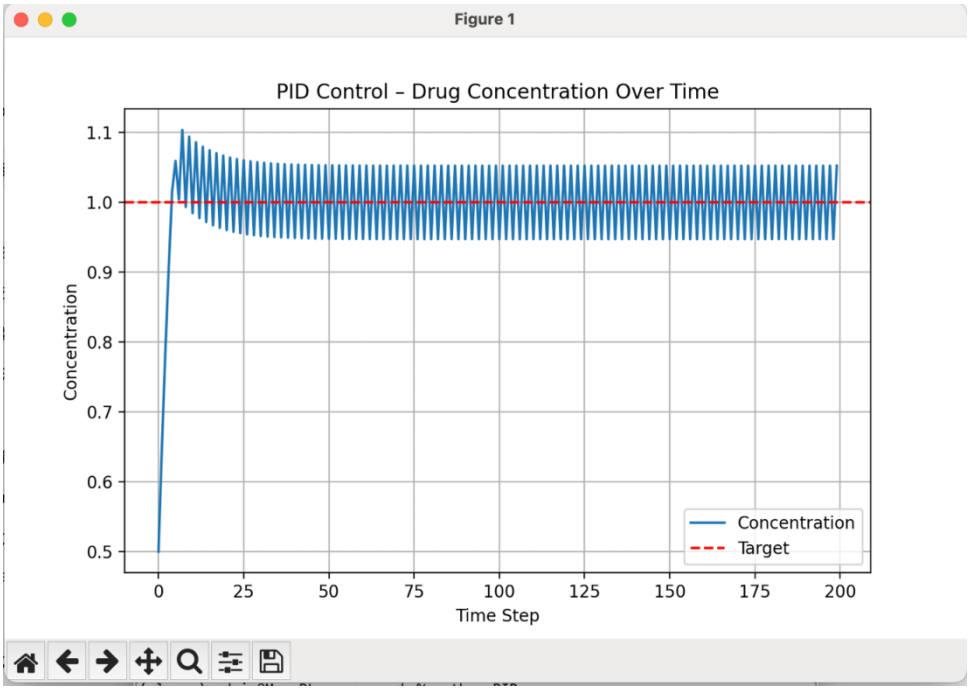

Figure 9. PID concentration tracking example (oscillation)

Figure 9 shows a representative example of PID control with an aggressive gain setting. The blood concentration reaches the target region relatively quickly, but the response exhibits noticeable oscillations and overshoot. The controller repeatedly over-corrects the error, leading to a series of damped waves around the target. While such behaviour may still be acceptable in some engineering systems, it can be undesirable in a clinical dosing scenario, where repeated overshoot might translate into transient toxicity or increased side-effect risk.

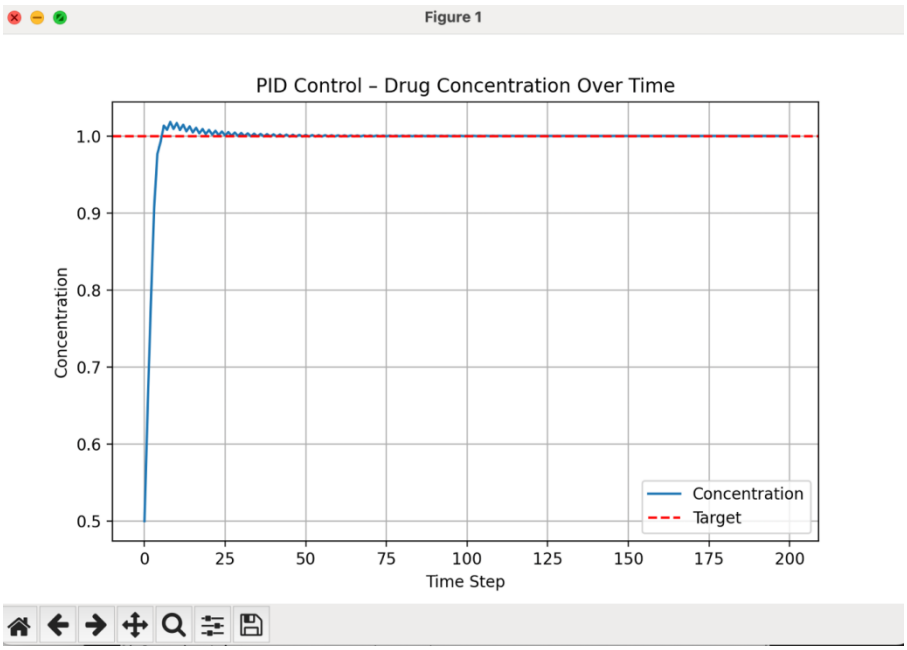

Figure 10. PID concentration tracking example (after tuning)

Figure 10 illustrates a PID controller after careful gain tuning. Compared with Figure 9, the oscillations are much smaller and the trajectory approaches the target more smoothly, demonstrating that a well-tuned PID can achieve satisfactory tracking. However, this improvement relies on manual parameter adjustment and is sensitive to changes in patient dynamics or operating conditions. In contrast, PPO learns a dosing policy directly from interaction data and can in principle adapt to different pharmacokinetic profiles without re-tuning explicit gains. This highlights the potential advantage of RL-based controllers in complex or time-varying bioengineering systems.

Table 4. PID vs PPO — Control Behaviour Comparison

Dimension	PID (Fig. 9 & 10)	PPO (Fig. 4–6)
Oscillation	Present, especially in Fig. 9	Reduced
Overshoot	Noticeable	Reduced
Parameter dependence	High	Low
Adaptability	Low	Higher
Stability	Only after tuning	Intrinsic

Qualitative Conclusion: PID can achieve good performance but requires manual tuning and is

sensitive to conditions, whereas PPO learns adaptively and offers more robust control.

5. Bioengineering Contextualisation

Reinforcement learning, particularly PPO, has great potential in bioengineering applications that require adaptive, real-time control. In this tutorial, we demonstrated PPO for regulating blood drug concentration in a simplified digital environment. This approach can be extended to real-world personalized drug delivery systems, where the goal is to maintain therapeutic drug levels within a patient's bloodstream while minimizing side effects. For instance, an RL agent could continuously adjust infusion rates in response to real-time sensor measurements, adapting to individual patient metabolism, circadian variations, or other physiological changes.

Beyond drug delivery, PPO and other RL methods could also be applied to robotic prosthetics: Adaptive control for movement assistance based on real-time biomechanical feedback. Rehabilitation robotics: Adjust exercise intensity or support in response to patient progress. Synthetic biology: Regulate gene expression or metabolic pathways in microbial cultures for optimized production. In summary, the PPO algorithm's ability to learn robust policies in continuous, dynamic environments makes it a promising tool for adaptive, personalized bioengineering solutions that require real-time decision-making.

I acknowledge the use of ChatGPT 5 (OpenAI, https://chat.openai.com/) to generate an outline for background study. I confirm that no content generated by AI has been presented as my own work

6. References

[1] Wang, X., Ma, Z., Cao, L. et al. A planar tracking strategy based on multiple-interpretable improved PPO algorithm with few-shot technique. Sci Rep 14, 3910 (2024). https://doi.org/10.1038/s41598-024-54268-6

[2] de Miguel Gomez, A. and Toosi, F. G. (2021). Continuous Parameter Control in Genetic Algorithms using Policy Gradient Reinforcement Learning. In Proceedings of the 13th

International Joint Conference on Computational Intelligence (IJCCI 2021) - ECTA; ISBN 978-989-758-534-0; ISSN 2184-3236, SciTePress, pages 115-122. DOI: 10.5220/0010643500003063

[3] Kargin, T.C., Kołota, J. A Reinforcement Learning Approach for Continuum Robot Control. J Intell Robot Syst 109, 77 (2023). https://doi.org/10.1007/s10846-023-02003-0

[4] D. Adamu Aliyu, E. Akashah Patah Akhir, M. Omar Abdullah Sawad, J. Shehu Yalli and Y. Saidu, "A Reinforcement Learning Approach to Personalized Asthma Exacerbation Prediction Using Proximal Policy Optimization," in IEEE Access, vol. 13, pp. 103373-103384, 2025, doi: 10.1109/ACCESS.2025.3579198.

[5] C. -H. Cho, P. -J. Huang, M. -C. Chen and C. -W. Lin, "Closed-Loop Deep Brain Stimulation With Reinforcement Learning and Neural Simulation," in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 32, pp. 3615-3624, 2024, doi: 10.1109/TNSRE.2024.3465243.

[6] D. F. G. Filho, M. A. Moret and E. G. S. Nascimento, "A Custom Reinforcement Learning Environment for Hybrid Renewable Energy Systems: Design and Implementation," in IEEE Access, vol. 13, pp. 133984-133993, 2025, doi: 10.1109/ACCESS.2025.3593064.

[7] Tan, C.; Wang, J.; Cai, H.; Hu, S.; Zhang, B.; Zhu, W. Phase-Adaptive Reinforcement Learning for Self-Tuning PID Control of Cruise Missiles. Aerospace 2025, 12, 849. https://doi.org/10.3390/aerospace12090849

[8] Jifei Deng, Seppo Sierla, Jie Sun, Valeriy Vyatkin, Reinforcement learning for industrial process control: A case study in flatness control in steel industry, Computers in Industry, 143, 2022, 103748, ISSN 0166-3615, https://doi.org/10.1016/j.compind.2022.103748.

[9] Garcia, C. (2024). Reinforcement learning for dynamic pricing and capacity allocation in monetized customer wait-skipping services. Journal of Business Analytics, 8(1), 36–54. https://doi.org/10.1080/2573234X.2024.2424542

[10] F. Rahimi Ghashghaei, N. Elmrabit, A. -U. -H. Qureshi, A. Akhunzada and M. Yousefi, "Advanced Quantum Control With Ensemble Reinforcement Learning: A Case Study on the

XY Spin Chain," in IEEE Access, vol. 13, pp. 49514-49526, 2025, doi:
10.1109/ACCESS.2025.3551232