Lake_1

Lake problem describes a situation where inhabitants of a town intend to increase their economic benefits through developing industry and agriculture. These activities will also emit pollution into a lake nearby. Although the lake pollution decreases continuously due to natural removal, once it exceeds a threshold X_{crit}, irreversible lake eutrophication would occur and cause huge losses. This problem requires decision-makers to decide the annual pollution emissions, with the aim of maximizing the economic benefit while avoiding eutrophication. The lake pollution transition function is given by
\begin{equation}
    X_{(t+1)}=X_t+a_t+\frac{(X_t^q)}{(1+X_t^q)}- bX_t+\epsilon_t
\end{equation}
where X_t is the pollution at time t, and $X_0 = 0$; a_t is the rate of anthropogenic pollution at time t; b is the natural removal rate of the pollution; q is the natural recycling rate of the pollution; ϵ_t is the rate of natural pollution at time t. The threshold X_{crit} is the pollution level at which the lake's natural recycling equals its natural removal rate. The total benefit from the pollution is given by
\begin{equation}
    f_{economic} = \sum\limits_{t \in {T}}\alpha a_t \delta^t
\end{equation}
where α is the benefit-to-pollution ratio, and δ is the discount rate.

The lake problem is characterized by parameter uncertainty. The natural pollution ϵ_t follows a log-normal distribution with mean μ and variance σ^2. In contrast, the probability distributions of the parameters μ, σ, b, q, δ cannot be determined.

Please formalize this problem description as a simulation model and specify the key components, including state variables, decision variables, transition functions, objective functions, stochastic variables, constant parameters and deep uncertainties.

Below is a clean **formalization of the Lake Problem as a simulation / decision model**, separating deterministic dynamics, stochastic elements, and deep uncertainty. I present it in a structure commonly used in **dynamic decision-making, control, and robust policy analysis**.

---

1. Model Structure Overview

The lake problem can be formalized as a **discrete-time stochastic dynamic system** with:

- A single state variable (lake pollution)
- A control (decision) variable (anthropogenic emissions)
- Nonlinear system dynamics
- An economic objective with discounting

- A hard ecological constraint (irreversible eutrophication)
- Both **aleatory uncertainty** (stochastic pollution shocks) and **deep uncertainty** (unknown parameter distributions)

2. Time Index

- Time horizon:

$$t = 0, 1, 2, \ldots, T$$

3. State Variable

- **Lake pollution stock**

$$X_t \in \mathbb{R}_{\geq 0}$$

- Initial condition:

$$X_0 = 0$$

4. Decision Variable (Control)

- **Anthropogenic pollution emissions**

$$a_t \in \mathbb{R}_{\geq 0}$$

- Chosen by decision-makers at each time step t.

5. Transition (State Dynamics) Function

The evolution of lake pollution is governed by:

$$X_{t+1} = g(X_t, a_t, \epsilon_t)$$

where:

$$g(X_t, a_t, \epsilon_t) = X_t + a_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

Interpretation of Terms

- a_t: controllable anthropogenic input
- $\frac{X_t^q}{1+X_t^q}$: nonlinear natural recycling (positive feedback)
- bX_t: natural removal
- ϵ_t: stochastic natural pollution inflow

6. Ecological Threshold (Constraint)

- **Critical pollution level**:

$$X_{\text{crit}} \text{ such that } \frac{X_{\text{crit}}^q}{1+X_{\text{crit}}^q} = bX_{\text{crit}}$$

- **Irreversibility constraint**:

$$X_t \geq X_{\text{crit}} \;\Rightarrow\; \text{lake eutrophication (absorbing failure state)}$$

Once exceeded, the system transitions into a permanently degraded state with very large or infinite loss.

7. Objective Function

The goal is to **maximize discounted economic benefits** from pollution emissions:

$$\max_{\{a_t\}_{t=0}^T} \quad \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t \delta^t\right]$$

where:

- $\alpha > 0$: benefit-to-pollution ratio
- $\delta \in (0, 1)$: discount factor
- Expectation is taken over the stochastic process $\{\epsilon_t\}$

Optionally, the objective may be written as a **constrained optimization**:

$$\max \; \mathbb{E}[f_{economic}] \quad \text{s.t.} \quad \Pr(X_t < X_{\text{crit}} \; \forall t) \geq \theta$$

for some safety threshold θ.

8. Stochastic Variables (Aleatory Uncertainty)

Natural Pollution

$$\epsilon_t \sim \text{LogNormal}(\mu, \sigma^2)$$

- Independent and identically distributed across time
- Mean: μ
- Variance: σ^2

This represents **environmental variability** that cannot be controlled.

9. Constant Parameters (Known or Assumed Fixed)

Parameter	Description
α	Economic benefit per unit pollution
T	Planning horizon
X_{crit}	Pollution threshold for eutrophication

10. Deep Uncertainty (Epistemic Uncertainty)

The following parameters are **deeply uncertain**, meaning their probability distributions are unknown or contested:

- μ: mean of natural pollution
- σ: variance of natural pollution
- b: natural removal rate
- q: recycling nonlinearity parameter
- δ: discount factor

Formally:

$$(\mu, \sigma, b, q, \delta) \in \Theta$$

where Θ is an **uncertain but bounded parameter set**, not a known probability distribution.

11. Simulation Model Summary

The simulation model consists of:

1. **State update equation**

$$X_{t+1} = g(X_t, a_t, \epsilon_t)$$

2. **Decision rule (policy)**

$$a_t = \pi_t(X_t)$$

3. **Random sampling of ϵ_t**
4. **Failure condition**

$$X_t \geq X_{\text{crit}} \Rightarrow \text{terminal failure}$$

5. **Performance metrics**
   - Expected economic benefit
   - Probability of eutrophication
   - Time-to-failure (if any)

12. Interpretation

This is a **stochastic, nonlinear, dynamic decision problem with irreversible thresholds and deep uncertainty**, making it well-suited for:

- Robust Decision Making (RDM)
- Stochastic dynamic programming (with approximations)
- Adaptive policies
- Scenario discovery and stress testing

If you want, I can next:

- Express this as a **Markov Decision Process**
- Provide **pseudo-code for simulation**
- Reformulate it for **robust or adaptive policy optimization**

X_{crit} is not a constant because it depends on other deeply uncertain parameters. Please refine your model specification.

Good point — the threshold is **endogenous** and inherits deep uncertainty from the biophysical parameters. Below is a **refined and internally consistent model specification** that treats X_{crit} correctly.

---

1. Revised Role of the Critical Threshold

Definition (Endogenous Threshold)

The critical pollution level is **not a fixed constant**, but a function of uncertain ecological parameters:

$$X_{\text{crit}} = h(b, q)$$

where X_{crit} is defined implicitly as the **non-trivial equilibrium** at which:

$$\text{natural recycling} = \text{natural removal}$$

That is,

$$\frac{X_{\text{crit}}^q}{1 + X_{\text{crit}}^q} = bX_{\text{crit}} \tag{1}$$

For given (b, q), equation (1) may have:

- a trivial equilibrium $X = 0$
- one or more positive equilibria
  The smallest **unstable positive equilibrium** is taken as X_{crit}.

Thus, eutrophication risk depends directly on the uncertain parameters b and q.

---

2. Updated Model Components

2.1 Time Index

$$t = 0, 1, \ldots, T$$

2.2 State Variable

- Lake pollution stock:

$$X_t \geq 0, \quad X_0 = 0$$

2.3 Decision Variable

- Anthropogenic pollution emissions:

$$a_t \geq 0$$

Chosen at each time step based on available information.

2.4 Transition Function (Unchanged)

$$X_{t+1} = X_t + a_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

This defines a **nonlinear stochastic dynamical system** parameterized by (b, q).

---

3. Stochastic Variables (Aleatory Uncertainty)

Natural Pollution Inflow

$$\epsilon_t \sim \text{LogNormal}(\mu, \sigma^2), \quad \text{i.i.d. over } t$$

---

4. Deeply Uncertain Parameters

The following parameters are **deeply uncertain**, with unknown or disputed probability distributions:

$$\theta = (\mu, \sigma, b, q, \delta)$$

They belong to an uncertainty set:

$$\theta \in \Theta$$

No probabilistic weighting over Θ is assumed.

---

5. Endogenous Failure Condition

Parameter-Dependent Failure Set

For a given realization of (b, q), define:

$$\mathcal{F}(b, q) = \{X_t : X_t \geq X_{\text{crit}}(b, q)\}$$

where $X_{\text{crit}}(b, q)$ satisfies equation (1).

Irreversibility

Once the system enters $\mathcal{F}(b, q)$, the lake transitions to an **absorbing eutrophic state**, with catastrophic loss.

---

6. Objective Function (Refined)

The planner seeks to maximize expected discounted benefits **while avoiding failure across deep uncertainties**:

$$\max_{\pi} \; \mathbb{E}_{\epsilon} \left[\sum_{t=0}^{T} \alpha a_t \delta^t \, \middle| \, \theta \right]$$

subject to:

$$a_t = \pi_t(X_t)$$

and evaluated across:

$$\theta \in \Theta$$

---

7. Robust / Safety-Oriented Performance Metrics

Because X_{crit} is uncertain, performance is assessed using **robust criteria**, for example:

(a) Worst-Case Constraint

$$\sup_{\theta \in \Theta} \Pr_{\epsilon} \left(\exists t : X_t \geq X_{\text{crit}}(b, q) \right) \leq \eta$$

(b) Regret-Based Objective

$$\min_{\pi} \sup_{\theta \in \Theta} \left[V^*(\theta) - V^{\pi}(\theta) \right]$$

(c) Satisficing / Safe Operating Space

$$X_t < X_{\text{crit}}(b, q) \quad \forall t, \; \forall \theta \in \Theta$$

---

8. Simulation Model (Final Form)

A single simulation run proceeds as:

1. Sample a parameter vector

$$\theta = (\mu, \sigma, b, q, \delta) \in \Theta$$

2. Compute $X_{\text{crit}}(b, q)$ by solving (1)
3. Initialize $X_0 = 0$
4. For $t = 0, \ldots, T$:
   - Choose $a_t = \pi_t(X_t)$
   - Sample $\epsilon_t \sim \text{LogNormal}(\mu, \sigma^2)$

- Update X_{t+1}
- Check failure: $X_{t+1} \geq X_{\text{crit}}(b, q)$

5. Record:
   - Economic benefits
   - Failure occurrence and timing

---

9. Key Conceptual Refinement

Crucially:

- The lake does **not** have one threshold
- Each plausible ecological reality (b, q) induces its **own tipping point**
- Policies must therefore be **robust to a family of tipping points**, not a single known boundary

This makes the lake problem a canonical example of a **deeply uncertain tipping-point control problem**.

If you want, I can next:

- Express this as a **parameterized MDP**
- Show how this is implemented in **exploratory modeling / RDM**
- Derive the **bifurcation structure** of the recycling–removal balance

Various stakeholders approach this problem from different perspectives. Please refine the model specification from the perspectives of (1) the local community and (2) the environmental regulator. These two perspectives should share a common environment, with each stakeholder independently controlling its own decision variables. In the specification, specify the common environment, including the global parameters and variables. For each perspective, also specify the corresponding model and its key components, including the decision variables, transition functions, and objective functions. If there are multiple objectives, provide multiple objective functions instead of a weighted objective function.

Below is a **two-stakeholder refinement** of the lake problem as a **shared-environment, multi-actor simulation model**. The local community and the environmental regulator interact through the same lake dynamics but **control different decision variables and pursue different objectives**, without collapsing objectives into weighted sums.

I. Common Environment (Shared by All Stakeholders)

This layer defines the **physical lake system, uncertainties, and shared state variables**. Neither stakeholder controls this layer directly.

I.1 Time

$$t = 0, 1, \ldots, T$$

I.2 Global State Variables

- **Lake pollution stock**

$$X_t \in \mathbb{R}_{\geq 0}, \quad X_0 = 0$$

- **Eutrophication indicator**

$$E_t = \begin{cases} 0 & X_t < X_{\text{crit}}(b, q) \\ 1 & X_t \geq X_{\text{crit}}(b, q) \end{cases}$$

Once $E_t = 1$, the system enters an **absorbing degraded state**.

I.3 Global Transition Function (Environment Dynamics)

$$X_{t+1} = X_t + A_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

where:

- A_t = **total anthropogenic pollution** (aggregated from agents)
- ϵ_t = stochastic natural pollution

I.4 Stochastic Process (Aleatory Uncertainty)

$$\epsilon_t \sim \text{LogNormal}(\mu, \sigma^2), \quad \text{i.i.d.}$$

I.5 Deeply Uncertain Parameters (Global)

$$\theta = (\mu, \sigma, b, q, \delta) \in \Theta$$

No probability distribution over Θ is assumed.

I.6 Endogenous Critical Threshold (Global)

$$X_{\text{crit}}(b, q) \;\; \text{solves} \;\; \frac{X^q}{1 + X^q} = bX$$

This threshold is **shared** and **parameter-dependent**, but unknown ex ante.

II. Stakeholder 1: Local Community Perspective

The local community represents **industry and agriculture** seeking economic benefits from pollution-generating activities.

II.1 Community Decision Variables

- **Anthropogenic emissions**

$$a_t^C \geq 0$$

II.2 Information Available to the Community

- Observes current pollution level X_t
- Does **not know** the true values of (b, q, μ, σ)
- May or may not anticipate regulatory actions

II.3 Community Contribution to Environment

$$A_t = a_t^C \quad \text{(before regulation)}$$

or, if regulation applies,

$$A_t = a_t^C - r_t$$

where r_t is imposed externally by the regulator.

II.4 Community-Controlled State Transition (Induced)

The community does **not** control the full transition, but its action affects it via a_t^C:

$$X_{t+1} = X_t + a_t^C + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

II.5 Community Objectives (Multiple, Non-Aggregated)

Objective C1: Maximize Economic Benefits

$$\max_{\pi^C} \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t^C \delta^t\right]$$

Objective C2: Avoid Economic Collapse from Eutrophication

$$\min_{\pi^C} \Pr\left(\exists t : X_t \geq X_{\text{crit}}(b, q)\right)$$

Objective C3 (Optional): Stability of Production

$$\min_{\pi^C} \mathrm{Var}(a_t^C)$$

This reflects political and economic resistance to volatile production levels.

II.6 Community Policy Form

$$a_t^C = \pi_t^C(X_t)$$

Possibly adaptive, reactive, or myopic.

III. Stakeholder 2: Environmental Regulator Perspective

The regulator represents a public authority tasked with **protecting the lake**, potentially at the cost of reduced economic output.

III.1 Regulator Decision Variables

The regulator does **not** emit pollution. Instead, it controls constraints on emissions.

Examples:

Option R1: Emission Cap

$$a_t^C \leq \bar{a}_t$$

Option R2: Pollution Abatement / Removal

$$r_t \geq 0 \quad \Rightarrow \quad A_t = a_t^C - r_t$$

Option R3: Pollution Tax (implicit control)

$$\tau_t \geq 0 \quad \text{(affects community behavior)}$$

III.2 Regulator-Controlled State Transition (Induced)

With abatement:

$$X_{t+1} = X_t + (a_t^C - r_t) + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

III.3 Regulator Objectives (Multiple, Non-Weighted)

Objective R1: Prevent Eutrophication

$$\min_{\pi^R} \Pr\left(\exists t : X_t \geq X_{\text{crit}}(b, q)\right)$$

Objective R2: Minimize Expected Pollution Levels

$$\min_{\pi^R} \mathbb{E}\left[\sum_{t=0}^{T} X_t\right]$$

Objective R3: Minimize Regulatory Intervention Cost

$$\min_{\pi^R} \mathbb{E}\left[\sum_{t=0}^{T} c(r_t)\right]$$

where $c(\cdot)$ is an increasing convex cost function.

III.4 Regulator Policy Form

$$r_t = \pi_t^R(X_t) \quad \text{or} \quad \bar{a}_t = \pi_t^R(X_t)$$

Typically conservative and precautionary under deep uncertainty.

IV. Coupled Multi-Actor System Summary

Component	Description
Shared state	X_t, E_t
Shared dynamics	Nonlinear stochastic lake model
Community control	Pollution emissions a_t^C
Regulator control	Caps, abatement, or enforcement
Interaction	Regulator constrains community action
Uncertainty	Both stochastic (ϵ_t) and deep (θ)
Failure	Parameter-dependent tipping point

V. Conceptual Interpretation

- This is a **multi-objective, multi-agent dynamic system**
- Stakeholders interact **only through the shared environment**
- No stakeholder controls the full system
- Tradeoffs emerge endogenously rather than via weighted objectives
- The problem naturally supports:
  - Game-theoretic analysis
  - Adaptive governance
  - Robust and precautionary policy design

If you want, I can next:

- Cast this as a **dynamic game**
- Show **Pareto fronts** for regulator vs community
- Specify **adaptive trigger-based policies**
- Translate this into an **agent-based simulation architecture**

To ensure internal consistency across these models, please compose them into a single, unified model that shares a common environment and states, while embedding perspective-specific decisions and objectives. The composed model should: 1. embed each perspective's decision variables and objectives; 2. enable modular evaluation of each perspective; 3. for each perspective, treat other perspectives' decisions as exogenous inputs; and 4. incorporate all other components from every perspective.

Below is a **single, unified composed model** that embeds both stakeholder perspectives into **one internally consistent simulation architecture**, while preserving **modularity, separability of objectives, and exogeneity of other actors' decisions** for evaluation.

The model is written in a way that can be implemented directly as a **multi-actor simulation / dynamic game / exploratory model**.

Unified Composed Lake Model

1. Global Model Structure

The system is a **shared stochastic dynamical environment** with:

- One common state space
- One common transition function
- Multiple decision-makers
- Perspective-specific objectives
- Deep uncertainty affecting all components

Each stakeholder observes the same environment but **optimizes its own objectives**, treating the other's actions as **exogenous inputs**.

---

2. Common Environment (Shared by All Perspectives)

2.1 Time

$$t = 0, 1, \dots, T$$

2.2 Global State Variables

- **Lake pollution stock**

$$X_t \in \mathbb{R}_{\geq 0}, \quad X_0 = 0$$

- **Eutrophication indicator (absorbing)**

$$E_t = \begin{cases} 0 & X_t < X_{\text{crit}}(b, q) \\ 1 & X_t \geq X_{\text{crit}}(b, q) \end{cases}$$

Once $E_t = 1$, the system remains in the eutrophic state.

2.3 Global Stochastic Variable

- **Natural pollution inflow**

$$\epsilon_t \sim \text{LogNormal}(\mu, \sigma^2), \quad \text{i.i.d.}$$

2.4 Deeply Uncertain Global Parameters

$$\theta = (\mu, \sigma, b, q, \delta) \in \Theta$$

No probability distribution over Θ is assumed.

2.5 Endogenous, Parameter-Dependent Threshold

The critical pollution level is **not a constant**, but a function of uncertain ecological parameters:

$$X_{\text{crit}}(b, q) \quad \text{satisfying} \quad \frac{X^q}{1 + X^q} = bX$$

This function is computed separately for each realization of (b, q).

2.6 Global Transition Function

The shared lake dynamics are:

$$X_{t+1} = X_t + A_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

where:

$$A_t = a_t^C - r_t^R$$

is the **net anthropogenic pollution**, aggregated from stakeholder decisions.

3. Stakeholder-Specific Embedded Models

Each stakeholder model is defined as a **projection** of the global model, consisting of:

- Its own decision variables
- Its own policy
- Its own objectives
- The shared environment treated as given

4. Stakeholder 1: Local Community Module

4.1 Community Decision Variables

- **Emissions decision**

$$a_t^C \geq 0$$

4.2 Community Policy

$$a_t^C = \pi_t^C(X_t)$$

The community observes the shared state X_t and chooses emissions.

4.3 Community-Induced State Evolution

From the community's perspective, the state evolves as:

$$X_{t+1} = X_t + a_t^C - \underbrace{r_t^R}_{\text{exogenous}} + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

where the regulator's action r_t^R is treated as **exogenous**.

4.4 Community Objectives (Non-Aggregated)

The community evaluates outcomes using the following objectives:

C1. Economic benefit

$$\max \; J_C^{(1)} = \mathbb{E}\left[\sum_{t=0}^{T} \alpha a_t^C \delta^t\right]$$

C2. Avoidance of eutrophication

$$\min \; J_C^{(2)} = \Pr\left(\exists t : X_t \geq X_{\text{crit}}(b, q)\right)$$

(Optional additional objectives such as stability can be added without changing the environment.)

5. Stakeholder 2: Environmental Regulator Module

5.1 Regulator Decision Variables

- **Abatement / removal effort**

$$r_t^R \geq 0$$

(Other instruments such as caps or taxes can be represented equivalently.)

5.2 Regulator Policy

$$r_t^R = \pi_t^R(X_t)$$

5.3 Regulator-Induced State Evolution

From the regulator's perspective:

$$X_{t+1} = X_t + \underbrace{a_t^C}_{\text{exogenous}} - r_t^R + \frac{X_t^q}{1 + X_t^q} - bX_t + \epsilon_t$$

The community's emissions a_t^C are treated as **exogenous inputs**.

5.4 Regulator Objectives (Non-Aggregated)

R1. Prevent eutrophication

$$\min \; J_R^{(1)} = \Pr\left(\exists t : X_t \geq X_{\text{crit}}(b, q)\right)$$

R2. Minimize pollution levels

$$\min \; J_R^{(2)} = \mathbb{E}\left[\sum_{t=0}^{T} X_t\right]$$

R3. Minimize regulatory cost

$$\min \; J_R^{(3)} = \mathbb{E}\left[\sum_{t=0}^{T} c(r_t^R)\right]$$

6. Composition Logic (Key Requirement Satisfaction)

Requirement 1: Embed each perspective's decisions and objectives

✔ Each stakeholder has its own decision variables, policies, and objective vector.

Requirement 2: Enable modular evaluation

✔ Each module evaluates performance using its own objective functions while sharing the same simulated trajectory $\{X_t\}$.

Requirement 3: Treat other perspectives' decisions as exogenous

✔ In each stakeholder's induced transition function, the other actor's decisions appear explicitly as exogenous inputs.

Requirement 4: Incorporate all components

✔️ The composed model includes:

- Nonlinear lake dynamics
- Stochastic inflows
- Deep uncertainty
- Endogenous tipping points
- Irreversible failure
- Multi-objective evaluation

---

7. Unified Simulation Algorithm

For a given realization of $\theta \in \Theta$:

1. Compute $X_{\text{crit}}(b, q)$
2. Initialize $X_0 = 0$
3. For $t = 0, \ldots, T$:
    - Community chooses $a_t^C = \pi_t^C(X_t)$
    - Regulator chooses $r_t^R = \pi_t^R(X_t)$
    - Sample ϵ_t
    - Update X_{t+1}
    - Check eutrophication condition
4. Evaluate $\{J_C^{(i)}\}$ and $\{J_R^{(j)}\}$ separately

---

8. Conceptual Interpretation

This unified formulation is:

- A **multi-actor stochastic dynamic system**
- Equivalent to a **dynamic game with deep uncertainty**
- Suitable for:
    - Robust decision making (RDM)
    - Adaptive governance analysis
    - Pareto tradeoff exploration
    - Policy stress testing across tipping points

If you want, I can next:

- Formalize this as a **dynamic game with equilibrium concepts**
- Show how to implement this in **exploratory modeling frameworks**

- Define **adaptive trigger-based policies** for each stakeholder

Now, provide a modular Python implementation of the unified model, using classes to represent different perspectives. At the end of the program, define a function as the interface. This function should take as inputs a dictionary of decision variable values, a dictionary of uncertain parameter values, and a dictionary of constant parameter values; simulate the model dynamics; and return the value of each objective function. This function does not need to be executed.

Below is a **clean, modular Python implementation** of the unified model.
It uses **classes to represent the shared environment and each stakeholder perspective**, and ends with a **single interface function** that matches your specification.

The code is written for **clarity and extensibility**, not execution efficiency.

Design Overview (Mapping to the Unified Model)

- **`LakeEnvironment`**
  Shared environment: states, dynamics, stochastic inflow, eutrophication logic
- **`LocalCommunity`**
  Perspective-specific decisions and objectives
- **`EnvironmentalRegulator`**
  Perspective-specific decisions and objectives
- **`simulate_lake_model(...)`**
  Interface function:
  - Inputs:
    - decision variables (time series)
    - uncertain parameters
    - constant parameters
  - Outputs:
    - values of **each objective function**, separated by perspective

---

Python Implementation

```python
import numpy as np
from typing import Dict, List, Any
```

---

1. Shared Environment

python

```python
class LakeEnvironment:
    """
    Shared lake environment with nonlinear stochastic dynamics
    and endogenous eutrophication threshold.
    """

    def __init__(self, uncertain_params: Dict[str, float], constants: Dict[str, float]):
        self.mu = uncertain_params["mu"]
        self.sigma = uncertain_params["sigma"]
        self.b = uncertain_params["b"]
        self.q = uncertain_params["q"]

        self.T = constants["T"]

        self.X_crit = self._compute_critical_threshold()

    def _compute_critical_threshold(self) -> float:
        """
        Solve for X_crit from:
            X^q / (1 + X^q) = b X
        Uses a simple numerical search over a bounded domain.
        """

        X_grid = np.linspace(1e-6, 10.0, 10_000)
        lhs = X_grid**self.q / (1 + X_grid**self.q)
        rhs = self.b * X_grid

        diff = np.abs(lhs - rhs)
        return X_grid[np.argmin(diff)]

    def sample_natural_pollution(self) -> float:
        """
        Sample stochastic natural pollution inflow.
        """
        return np.random.lognormal(mean=self.mu, sigma=self.sigma)

    def step(self, X_t: float, a_c: float, r_r: float, epsilon_t: float) -> float:
        """
        One-step lake pollution dynamics.
        """

        net_anthropogenic = a_c - r_r
        recycling = X_t**self.q / (1 + X_t**self.q)
        removal = self.b * X_t

        X_next = X_t + net_anthropogenic + recycling - removal + epsilon_t
        return max(X_next, 0.0)
```

2. Local Community Perspective

python

```python
class LocalCommunity:
    """
    Local community controlling pollution emissions.
    """

    def __init__(self, constants: Dict[str, float]):
        self.alpha = constants["alpha"]
        self.delta = constants["delta"]

        self.economic_benefit = 0.0
        self.failed = False

    def reset(self):
        self.economic_benefit = 0.0
        self.failed = False
```

```
    def observe_and_act(self, t: int, X_t: float, decisions: Dict[str, List[float]]) -> float:
        """
        Return community emissions at time t.
        """
        return decisions["a_community"][t]

    def update_objectives(self, t: int, a_c: float):
        """
        Update economic objective.
        """
        self.economic_benefit += self.alpha * a_c * (self.delta ** t)

    def check_failure(self, X_t: float, X_crit: float):
        if X_t >= X_crit:
            self.failed = True
```

---

3. Environmental Regulator Perspective

```python
class EnvironmentalRegulator:
    """
    Environmental regulator controlling abatement.
    """

    def __init__(self):
        self.total_pollution = 0.0
        self.regulation_cost = 0.0
        self.failed = False

    def reset(self):
        self.total_pollution = 0.0
        self.regulation_cost = 0.0
        self.failed = False

    def observe_and_act(self, t: int, X_t: float, decisions: Dict[str, List[float]]) -> float:
        """
        Return regulator abatement at time t.
        """
        return decisions["r_regulator"][t]

    def update_objectives(self, X_t: float, r_r: float):
        self.total_pollution += X_t
        self.regulation_cost += r_r**2  # convex cost example

    def check_failure(self, X_t: float, X_crit: float):
        if X_t >= X_crit:
            self.failed = True
```

---

4. Unified Simulation Interface (Required Function)

```python
def simulate_lake_model(
    decisions: Dict[str, List[float]],
    uncertain_params: Dict[str, float],
    constants: Dict[str, float],
) -> Dict[str, Any]:
    """
    Unified interface for the composed lake model.

    Inputs
    ------
    decisions:
        {
```

```
            "a_community": [a_0, a_1, ..., a_T],
            "r_regulator": [r_0, r_1, ..., r_T]
        }

    uncertain_params:
        {
            "mu": float,
            "sigma": float,
            "b": float,
            "q": float
        }

    constants:
        {
            "T": int,
            "alpha": float,
            "delta": float
        }

    Returns
    -------
    Dictionary containing values of each objective function.
    """

    # Initialize environment and agents
    env = LakeEnvironment(uncertain_params, constants)
    community = LocalCommunity(constants)
    regulator = EnvironmentalRegulator()

    community.reset()
    regulator.reset()

    # Initial state
    X_t = 0.0
    T = constants["T"]

    for t in range(T):
        # Agents act (treating each other as exogenous)
        a_c = community.observe_and_act(t, X_t, decisions)
        r_r = regulator.observe_and_act(t, X_t, decisions)

        # Sample stochastic inflow
        epsilon_t = env.sample_natural_pollution()

        # Environment transition
        X_t = env.step(X_t, a_c, r_r, epsilon_t)

        # Update objectives
        community.update_objectives(t, a_c)
        regulator.update_objectives(X_t, r_r)

        # Check eutrophication
        community.check_failure(X_t, env.X_crit)
        regulator.check_failure(X_t, env.X_crit)

        if X_t >= env.X_crit:
            break  # absorbing failure state

    return {
        "community": {
            "economic_benefit": community.economic_benefit,
            "eutrophication_occurred": community.failed,
        },
        "regulator": {
            "total_pollution": regulator.total_pollution,
            "regulation_cost": regulator.regulation_cost,
            "eutrophication_occurred": regulator.failed,
        },
        "environment": {
            "X_crit": env.X_crit,
            "final_pollution": X_t,
        },
    }
```

Key Properties of This Implementation

✔️ **Single unified environment**
✔️ **Perspective-specific decision variables and objectives**
✔️ **Other perspectives treated as exogenous inputs**
✔️ **Modular and extensible (easy to add new stakeholders)**
✔️ **Compatible with robust decision-making and exploratory modeling**

If you want, I can next:

- Add **adaptive policies instead of fixed decision sequences**
- Extend this to **multiple communities**
- Show how to use this in **robust policy search or RDM experiments**
- Convert this into a **Gym-style environment**

According to the specification, "delta" is an uncertain parameter instead of a constant. Please correct your implementation.

You're right — thank you for catching that. Below is a **corrected implementation** in which **`delta` is treated as a deeply uncertain parameter**, not a constant.

The fix involves:

1. Moving **`delta`** from **`constants`** to **`uncertain_params`**
2. Passing **`delta`** into the **`LocalCommunity`** from **`uncertain_params`**
3. Updating the interface documentation accordingly

No other model logic changes.

Corrected Modular Python Implementation

```python
import numpy as np
from typing import Dict, List, Any
```

---

1. Shared Environment (unchanged)

```python
class LakeEnvironment:
    """
    Shared lake environment with nonlinear stochastic dynamics
    and endogenous eutrophication threshold.
    """

    def __init__(self, uncertain_params: Dict[str, float], constants: Dict[str, float]):
        self.mu = uncertain_params["mu"]
        self.sigma = uncertain_params["sigma"]
        self.b = uncertain_params["b"]
        self.q = uncertain_params["q"]

        self.T = constants["T"]

        self.X_crit = self._compute_critical_threshold()

    def _compute_critical_threshold(self) -> float:
        """
        Solve X^q / (1 + X^q) = b X using a simple grid search.
        """
        X_grid = np.linspace(1e-6, 10.0, 10_000)
        lhs = X_grid**self.q / (1 + X_grid**self.q)
        rhs = self.b * X_grid
        diff = np.abs(lhs - rhs)
        return X_grid[np.argmin(diff)]

    def sample_natural_pollution(self) -> float:
```

```
        return np.random.lognormal(mean=self.mu, sigma=self.sigma)

    def step(self, X_t: float, a_c: float, r_r: float, epsilon_t: float) -> float:
        net_anthropogenic = a_c - r_r
        recycling = X_t**self.q / (1 + X_t**self.q)
        removal = self.b * X_t
        X_next = X_t + net_anthropogenic + recycling - removal + epsilon_t
        return max(X_next, 0.0)
```

2. Local Community (corrected: `delta` is uncertain)

```python
class LocalCommunity:
    """
    Local community controlling pollution emissions.
    """

    def __init__(self, uncertain_params: Dict[str, float], constants: Dict[str, float]):
        self.alpha = constants["alpha"]
        self.delta = uncertain_params["delta"]

        self.economic_benefit = 0.0
        self.failed = False

    def reset(self):
        self.economic_benefit = 0.0
        self.failed = False

    def observe_and_act(self, t: int, X_t: float, decisions: Dict[str, List[float]]) -> float:
        return decisions["a_community"][t]

    def update_objectives(self, t: int, a_c: float):
        self.economic_benefit += self.alpha * a_c * (self.delta ** t)

    def check_failure(self, X_t: float, X_crit: float):
        if X_t >= X_crit:
            self.failed = True
```

3. Environmental Regulator (unchanged)

```python
class EnvironmentalRegulator:
    """
    Environmental regulator controlling abatement.
    """

    def __init__(self):
        self.total_pollution = 0.0
        self.regulation_cost = 0.0
        self.failed = False

    def reset(self):
        self.total_pollution = 0.0
        self.regulation_cost = 0.0
        self.failed = False

    def observe_and_act(self, t: int, X_t: float, decisions: Dict[str, List[float]]) -> float:
        return decisions["r_regulator"][t]

    def update_objectives(self, X_t: float, r_r: float):
        self.total_pollution += X_t
        self.regulation_cost += r_r**2  # convex cost
```

```
    def check_failure(self, X_t: float, X_crit: float):
        if X_t >= X_crit:
            self.failed = True
```

---

4. Corrected Unified Interface Function

```python
def simulate_lake_model(
    decisions: Dict[str, List[float]],
    uncertain_params: Dict[str, float],
    constants: Dict[str, float],
) -> Dict[str, Any]:
    """
    Unified interface for the composed lake model.

    Parameters
    ----------
    decisions:
        {
            "a_community": [a_0, ..., a_T],
            "r_regulator": [r_0, ..., r_T]
        }

    uncertain_params:
        {
            "mu": float,
            "sigma": float,
            "b": float,
            "q": float,
            "delta": float
        }

    constants:
        {
            "T": int,
            "alpha": float
        }
    """

    env = LakeEnvironment(uncertain_params, constants)
    community = LocalCommunity(uncertain_params, constants)
    regulator = EnvironmentalRegulator()

    community.reset()
    regulator.reset()

    X_t = 0.0
    T = constants["T"]

    for t in range(T):
        a_c = community.observe_and_act(t, X_t, decisions)
        r_r = regulator.observe_and_act(t, X_t, decisions)

        epsilon_t = env.sample_natural_pollution()
        X_t = env.step(X_t, a_c, r_r, epsilon_t)

        community.update_objectives(t, a_c)
        regulator.update_objectives(X_t, r_r)

        community.check_failure(X_t, env.X_crit)
        regulator.check_failure(X_t, env.X_crit)

        if X_t >= env.X_crit:
            break  # absorbing eutrophic state

    return {
        "community": {
            "economic_benefit": community.economic_benefit,
```

```
            "eutrophication_occurred": community.failed,
        },
        "regulator": {
            "total_pollution": regulator.total_pollution,
            "regulation_cost": regulator.regulation_cost,
            "eutrophication_occurred": regulator.failed,
        },
        "environment": {
            "X_crit": env.X_crit,
            "final_pollution": X_t,
        },
    }
```

Summary of the Correction

- ✅ **`delta`** is now **fully part of deep uncertainty**
- ✅ Each simulation run evaluates objectives under a **specific realization of `delta`**
- ✅ The implementation is now **internally consistent** with the formal specification and stakeholder models

If you want, I can next:

- Add **heterogeneous discounting across stakeholders**
- Make **`delta` time-varying or ambiguous**
- Integrate this with **robust or exploratory policy search**

The possible value of X_{crit} is in the range [0.01, 1.5], and its exact value can be estimated using the root-finding function brentq. Please refine your implementation.

Below is a **refined and internally consistent correction** to the implementation that:

- Constrains X_{crit} to the range **[0.01, 1.5]**
- Computes X_{crit} using a **proper root-finding method (`brentq`)**
- Leaves all other components of the unified model unchanged

Only the **environment component** needs modification.

Key Refinement (Conceptual)

We now compute X_{crit} as the root of:

$$f(X) = \frac{X^q}{1 + X^q} - bX$$

on the bounded interval:

$$X \in [0.01, 1.5]$$

This is:

- **Numerically well-posed**
- **Consistent with the ecological interpretation**
- **Robust to deep uncertainty in b, q**

If no root exists in this interval, we explicitly flag that case.

Refined Implementation (Only the Changed Parts)

Required Import

```python
from scipy.optimize import brentq
```

Corrected `LakeEnvironment` Class

```python
class LakeEnvironment:
    """
    Shared lake environment with nonlinear stochastic dynamics
    and endogenous eutrophication threshold.
    """

    def __init__(self, uncertain_params: Dict[str, float], constants: Dict[str, float]):
        self.mu = uncertain_params["mu"]
        self.sigma = uncertain_params["sigma"]
        self.b = uncertain_params["b"]
        self.q = uncertain_params["q"]

        self.T = constants["T"]

        self.X_crit = self._compute_critical_threshold()

    def _threshold_equation(self, X: float) -> float:
        """
        Equation whose root defines X_crit:
            X^q / (1 + X^q) - b X = 0
        """
        return (X**self.q) / (1 + X**self.q) - self.b * X

    def _compute_critical_threshold(self) -> float:
        """
        Compute X_crit using Brent's method over [0.01, 1.5].
        """

        lower, upper = 0.01, 1.5

        f_lower = self._threshold_equation(lower)
        f_upper = self._threshold_equation(upper)

        if f_lower * f_upper > 0:
            raise ValueError(
                "No root for X_crit found in [0.01, 1.5] "
                f"for b={self.b}, q={self.q}"
            )

        return brentq(self._threshold_equation, lower, upper)

    def sample_natural_pollution(self) -> float:
        return np.random.lognormal(mean=self.mu, sigma=self.sigma)

    def step(self, X_t: float, a_c: float, r_r: float, epsilon_t: float) -> float:
        net_anthropogenic = a_c - r_r
        recycling = X_t**self.q / (1 + X_t**self.q)
        removal = self.b * X_t
        X_next = X_t + net_anthropogenic + recycling - removal + epsilon_t
        return max(X_next, 0.0)
```

Why This Refinement Matters

- ✔️ **Ecological realism**: avoids spurious equilibria at extreme pollution levels
- ✔️ **Numerical robustness**: Brent's method guarantees convergence if a root exists
- ✔️ **Deep uncertainty compatibility**: different (b, q) realizations yield different tipping points
- ✔️ **Explicit failure handling**: infeasible parameter combinations are detected, not hidden

---

Final Consistency Check

Requirement	Status
X_{crit} endogenous	✅
Bounded to [0.01, 1.5]	✅
Root-finding via **`brentq`**	✅
Shared across stakeholders	✅
Compatible with modular simulation	✅

If you want next, I can:

- Add **robust handling** when no root exists (e.g. precautionary defaults)
- Visualize how X_{crit} varies over (b, q)
- Extend the model to **multiple possible tipping points**
- Show how this integrates into **robust decision making (RDM)**

Your implementation of the lognormal distribution is incorrect. Please refine your implementation.

You're absolutely right — thank you for pointing this out.
The issue is **conceptual, not just syntactic**:

> In the problem specification,
> **μ and σ^2 are the mean and variance of the log-normal random variable ϵ_t**,
> **not** the mean and variance of the underlying normal distribution.

However, `np.random.lognormal(mean=…, sigma=…)` expects the **parameters of the underlying normal distribution**, usually denoted (μ_N, σ_N).

Below is a **correct and internally consistent refinement**.

---

1. Correct Lognormal Parameterization

Let:

$$\epsilon_t \sim \mathrm{LogNormal}(\mu, \sigma^2)$$

where:

- $\mu = \mathbb{E}[\epsilon_t]$
- $\sigma^2 = \mathrm{Var}(\epsilon_t)$

The corresponding parameters of the underlying normal distribution are:

$$\sigma_N^2 = \ln\left(1 + \frac{\sigma^2}{\mu^2}\right)$$

$$\mu_N = \ln(\mu) - \frac{1}{2}\sigma_N^2$$

These are the values that must be passed to `np.random.lognormal`.

---

2. Refined Implementation

Corrected `LakeEnvironment` (only changed parts shown)

```python
class LakeEnvironment:
    """
    Shared lake environment with nonlinear stochastic dynamics
    and endogenous eutrophication threshold.
    """
```

```
    def __init__(self, uncertain_params: Dict[str, float], constants: Dict[str, float]):
        self.mu_eps = uncertain_params["mu"]          # mean of epsilon_t
        self.var_eps = uncertain_params["sigma"]**2 # variance of epsilon_t
        self.b = uncertain_params["b"]
        self.q = uncertain_params["q"]

        self.T = constants["T"]

        # Convert to underlying normal parameters
        self._compute_lognormal_parameters()

        self.X_crit = self._compute_critical_threshold()

    def _compute_lognormal_parameters(self):
        """
        Convert (mean, variance) of lognormal distribution
        to parameters of underlying normal distribution.
        """
        self.sigma_N_sq = np.log(1.0 + self.var_eps / self.mu_eps**2)
        self.sigma_N = np.sqrt(self.sigma_N_sq)
        self.mu_N = np.log(self.mu_eps) - 0.5 * self.sigma_N_sq

    def sample_natural_pollution(self) -> float:
        """
        Sample epsilon_t from LogNormal(mu, sigma^2),
        where mu and sigma^2 are the mean and variance
        of the lognormal distribution.
        """
        return np.random.lognormal(mean=self.mu_N, sigma=self.sigma_N)
```

---

3. What Was Wrong Before (Explicitly)

❌ Previous (Incorrect)

```python
np.random.lognormal(mean=self.mu, sigma=self.sigma)
```

This incorrectly treated:

- `mu` as the **mean of the underlying normal**
- `sigma` as the **standard deviation of the underlying normal**

That contradicts the model specification.

---

4. Why This Correction Matters

Aspect	Impact
Statistical correctness	✔ Ensures correct moments of ϵ_t
Policy evaluation	✔ Prevents biased risk estimates
Tipping-point analysis	✔ Correct tail behavior near X_{crit}
Deep uncertainty	✔ Properly reflects ambiguity in natural inflows

An incorrect parameterization **systematically under- or over-estimates extreme pollution events**, which is especially dangerous in a **tipping-point problem**.

5. Final Consistency Check

Component	Status
Lognormal mean = μ	✅
Lognormal variance = σ^2	✅
Correct conversion to normal	✅
Shared across all perspectives	✅
Compatible with deep uncertainty	✅

If you want next, I can:

- Add **input validation** to ensure $\mu > 0$
- Make ϵ_t **temporally correlated**
- Show analytically how tail risk affects eutrophication probability
- Extend the model to **multiple stochastic inflows**