BSc Thesis

Valdemar H. Lorenzen

*Melih Kandemir

IMADA, SDU

Abstract

TODO

Write the abstract

1 Introduction

Many real-world applications present an inherent challenge that current reinforcement learning (RL) methods struggle to address effectively: the problem of delayed and sparse rewards [1], [2].

> **Delayed and Sparse Rewards:** Learning scenarios where meaningful feedback signals (rewards) are provided only far after a long sequence of actions, and where most actions yield no immediate feedback.
>
> *Example: In drug discovery, the effectiveness of a designed molecule can only be evaluated after its complete synthesis, with no intermediate feedback during the design process.*

Consider, for instance, the process of drug design, where a reinforcement learning agent must make a series of molecular modifications to create an effective compound. The value of these decisions — the drug's efficacy — can only be assessed once the entire molecule is complete. Similarly, in robotics tasks like assembly or navigation, success often depends on precise sequences of actions where feedback is only available upon having completed the entire task.

Traditional reinforcement learning algorithms face two critical limitations in such environments:

1. **Credit Assignment:** When rewards are delayed, the algorithm struggles to correctly attribute success or failure to specific actions in a long sequence [3]. This is analogous to trying to improve a chess strategy when only knowing the game's outcome, without understanding which moves were actually decisive.
2. **Exploration Efficiency:** With sparse rewards, random exploration becomes highly inefficient [4], [5]. An agent might need to execute precisely the right sequence of actions to receive any feedback at all, making random exploration about as effective as searching for a needle in a haystack.

This thesis investigates a novel approach to addressing these challenges through the comparison of two promising methodologies: **Generative Flow Networks** (GFlowNets) as proposed by [6], and **Bayesian Exploration Networks** (BEN) as proposed by [7]. These approaches represent different perspectives on handling uncertainty and exploration in reinforcement learning.

1. *GFlowNets* frame the learning process as a flow network, potentially offering more robust learning in situations with multiple viable solutions.
2. *BENs* leverages Bayesian uncertainty estimation to guide exploration more efficiently, potentially making better use of limited feedback.

By comparing these approaches, we aim to understand their relative strengths and limitations in environments with delayed and sparse rewards. Our investigation focuses specifically on examining these methods in carefully designed environments that capture the essential characteristics of delayed and sparse reward scenarios while remaining tractable for systematic analysis.

1.1 Research Objectives and Contributions

This thesis aims to advance our understanding of efficient learning in sparse reward environments through three primary objectives:

1. **Comparative Analysis:** Conduct a rigorous empirical comparison between GFlowNets and Bayesian Exploration Networks in standardized environments with delayed rewards.
2. **Hypothesis Testing:** Investigate whether BEN's Bayesian exploration strategy leads to more efficient learning compared to GFlowNets in highly delayed reward scenarios, particularly during early training stages.
3. **Algorithmic Understanding:** Analyze the underlying mechanisms that drive performance differences between these approaches, focusing on their handling of uncertainty and exploration.

The contributions of this work include:

- A comprehensive empirical evaluation using the n-chain environment with varying degrees of reward delay.
- Insights into the relative strengths and limitations of Bayesian and flow-based approaches to exploration.
- Implementation and analysis of both algorithms with comparisons.

1.2 Thesis and Structure

The remainder of this thesis is structured as follows:

Section 2: Preliminaries provides the theoretical foundations of reinforcement learning and explores existing approaches to handling sparse rewards. This chapter establishes the mathematical framework and notation used throughout the thesis.

Section 3: Theoretical Framework presents our hypothesis and analytical approach. We develop the mathematical foundations for comparing GFlowNets and BEN.

Section 4: Experimental Design details our testing methodology, including environment specifications, evaluation metrics, and implementation details.

Section 5: Results and Analysis presents our findings, including both quantitative performance metrics and qualitative analysis of learning behaviors. We examine how each algorithm handles the exploration-exploitation trade-off and adapts to varying levels of reward sparsity.

Section 6: Conclusion summarizes our findings, discusses their implications for the field, and suggests directions for future research.

2 Preliminaries

2.1 Flow Networks

GFlowNets rely on the concept of flow networks. The flow network is represented as a directed acyclic graph $G = (\mathcal{S}, \mathcal{A})$, where $\mathcal{S}$ represents the state space and $\mathcal{A}$ represents the action space.

> **Flow Network:** A directed acyclic graph with a single source node (initial state) and one or more sink nodes (terminal states), where flow is conserved at each intermediate node [6], [8].
>
> *Example: In molecular design, states represent partial molecules and actions represent adding molecular fragments.*

2.1.1 States and Trajectories

We distinguish severel types of states:

- An initial state $s_0 \in \mathcal{S}$ (the source);
- Terminal states $x \in \mathcal{X} \subset \mathcal{S}$ (sinks);
- Intermediate states that form the pathways from source to sinks.

A trajectory τ represents a complete path through the network, starting at s_0 and ending at some terminal state x. Formally, we write a trajectory as an ordered sequence $\tau = (s_0 \to s_1 \to \ldots \to s_n = x)$, where each transition $(s_t \to s_{t+1})$ corresponds to an action in $\mathcal{A}$.

2.1.2 Flow Function and Conservation

The *trajectory flow function* $F : \mathcal{T} \to \mathbb{R}_{\geq 0}$ assigns a non-negative value to each possible trajectory [8]. From this flow function, two important quantities are derived:

1. **State flow**: For any state s, its flow is the sum of flows through all trajectories passing through it:
$$F(s) = \sum_{s \in \tau} F(\tau). \tag{1}$$
2. **Edge flow**: For any action (edge) $s \to s'$, its flow is the sum of flows through all trajectories using that edge:
$$F(s \to s') = \sum_{\tau = (\ldots \to s \to s' \to \ldots)} F(\tau). \tag{2}$$

These flows must satisfy a conservation principle known as the *flow matching constraint*:

> **Flow Matching:** For any non-terminal state s, the total incoming flow must equal the total outgoing flow:
$$F(s) = \sum_{(s'' \to s) \in \mathcal{A}} F(s'' \to s) = \sum_{(s \to s') \in \mathcal{A}} F(s \to s'). \tag{3}$$

2.1.3 Markovian Flow

The flow function induces a probability distribution over trajectories. Given a flow function F, we define $P(\tau) = \frac{1}{Z}F(\tau)$ [8], where $Z = F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau)$ is the *partition function* — i.e., the total flow through the network.

> **Markovian Flow:** A flow is *Markovian* when it can be factored into local decisions at each state. This occurs when the following criteria are met [8]:
>
> 1. Forward policies $P_F(-|s)$ over children of each non-terminal state s.t.
>
> $$P(\tau = (s_0 \to ... \to s_n)) = \prod_{t=1}^{n} P_F(s_t|s_{t-1}). \tag{4}$$
>
> 2. Backward policies $P_B(-|s)$ over parents of each non-initial state s.t.
>
> $$P(\tau = (s_0 \to ... \to s_n)|s_n = x) = \prod_{t=1}^{n} P_B(s_{t-1}|s_t). \tag{5}$$

The Markovian property allows us to decompose complex trajectory distributions into simple local decisions, making learning tractable while maintaining the global flow constraints.

2.2 GFlowNets

GFlowNets are an approach to learning policies that sample from desired probability distributions [6]. They frame the learning process as discovering a flow function that makes the probability of generating any particular object proportional to its reward.

Given a reward function $R : \mathcal{X} \to \mathbb{R}_{\geq 0}$ defined over the set of terminal states $\mathcal{X}$, GFlowNets aim to approximate a Markovian flow F on the graph G s.t. $F(x) = R(x)$ for all $x \in \mathcal{X}$. We will make use of the following definition of a GFlowNet.

> **GFlowNet:** [8] defines a GFlowNet as any learning algorithm that discovers flow functions matching terminal state rewards, consisting of:
>
> 1. A model that outputs:
>   - Initial state flow $Z = F(s_0)$;
>   - Forward action distributions $P_F(-|s)$ for non-terminal states.
> 2. An objective function that, when globally minimized, guarantees $F(x) = R(x)$ for all terminal states.
>
> *Example: In molecular design, this ensures that high-reward molecules are genered more frequently, while maintaining diversity through exploration of multiple pathways.*

The power of GFlowNets lies in their ability to handle situations where multiple action sequences can lead to the same terminal state — a common scenario in real-world applications like molecular

design or image synthesis. Unlike traditional RL methods that focus on finding a single optimal path, GFlowNets learn a distribution over all possible paths directly proportional to their rewards.

2.2.1 Learning Process

The learning process of GFlowNets involves iteratively improving both flow estimates and the policies. The forward policy of a GFlowNet can sample trajectories from the learned Markovian flow F by sequentially selecting actions according to $P_F(-|s)$. When the training converges to a global minimum of the objective function, this sampling process guarantees that $P(x) \propto R(x)$. That is, the probability of generating any terminal state x is proportional to its reward $R(x)$. This property makes GFlowNets particularly well-suited for:

1. **Diverse Candidate Generation:** Rather than converging to a single solution, GFlowNets maintain a distribution over solutions weighted by their rewards.
2. **Multi-Modal Exploration:** The flow-based approach naturally handles problems with multiple distinct solutions of similar quality.
3. **Compositional-Structure Learning:** By learning flows over sequences of actions, GFlowNets can capture and generalize compositional patterns in the solution space.

To achieve this, GFlowNets employ various training objectives, with *trajectory balance* [8] being one such particularly effective objective.

2.2.2 Trajectory Balance

Trajectory balance focuses on ensuring consistency across entire trajectories, instead of matching flows at every state (which can be computationally expensive).

> **Trajectory Balance:** A principle that ensures the probability of generating a trajectory matches its reward by maintaining consistency between forward generation and backward reconstruction probabilities.

Consider a Markovian flow F that induces a distribution P over trajectories according to $P(\tau) = \frac{1}{Z}F(\tau)$. The forward policy P_F and backward policy P_B must satisfy the following *trajectory balance constraint* [8]

$$Z \prod_{t=1}^{n} P_F(s_t|s_{t-1}) = F(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t). \tag{6}$$

That is to say, the probability of constructing a trajectory forward should match the probability of reconstructing it backward, scaled by the appropriate rewards.

2.2.3 Trajectory Balance as an Objective

To convert the trajectory balance function into a training objective, we introduce a parametrized model with parameters θ that outputs:

1. A forward policy $P_F(-|s;\theta)$;

2. A backward policy $P_B(-|s;\theta)$;
3. A scalar estimate Z_θ of the partition function.

For any complete trajectory $\tau = (s_0 \to \ldots \to s_n = x)$, we define the *trajectory balance loss* as

$$\mathcal{L}_{\text{TB}}(\tau) = \left(\log \frac{Z_\theta \prod_{t=1}^{n} P_F(s_t|s_{t-1};\theta)}{R(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t;\theta)} \right)^2. \tag{7}$$

This loss captures how well our model satisfies the trajectory balance constraint. When the loss approaches zero, our model has learned to generate samples proportional to their rewards. In practice, we compute this loss in the log domain to avoid numerical stability, as suggested by [8]:

$$\mathcal{L}_{\text{TB}}(\tau) = \left(\log Z_\theta + \log \sum_{t=1}^{n} P_F(s_t|s_{t-1};\theta) - \log R(x) - \log \sum_{t=1}^{n} P_B(s_{t-1}|s_t;\theta) \right)^2. \tag{8}$$

[8] also remarks that a simplificatoin of Equation 7 occurs in tree-structured state spaces (when G is a directed tree), where each state has exactly one parent. In such cases, the backward policy becomes deterministic ($P_B = 1$), reducing the loss function to

$$\mathcal{L}_{\text{TB}}(\tau) = \left(\log \frac{Z_\theta \prod_{t=1}^{n} P_F(s_t|s_{t-1};\theta)}{R(x)} \right)^2, \tag{9}$$

which can be exploited for the n-chain environment.

The model is trained by sampling trajectories from a training policy π_θ — typically a tempered version of $P_F(-|-;\theta)$ to encourage exploration — and updating parameters using stochastic gradient descent: $\theta \leftarrow \theta - \alpha \mathbb{E}_{\tau\sim\pi_\theta} \nabla_\theta \mathcal{L}_{\text{TB}}(\tau)$.

2.3 Markov Decision Processes

The concept of the Markov Decision Process (MDP) [9] is fundamental in reinforcement learning and provides a model for sequential decision-making under uncertainty.

> **Markov Decision Process [9]:** A tuple $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, P_0, P_S, P_R, \gamma \rangle$ where:
>
> - $\mathcal{S}$ is the set of states;
> - $\mathcal{A}$ is the set of actions;
> - $P_0 \in \mathcal{P}(\mathcal{S})$ is the initial state distribution;
> - $P_S : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the state transition distribution;
> - $P_R : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$ is the reward distribution;
> - $\gamma \in [0, 1]$ is the discount factor.

At each timestep t, an agent observes its current state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ according to some policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. The environment then transitions to a new state s_{t+1} according

to the transition distribution $P_S(s_t, a_t)$ and provides a reward r_t sampled from $P_R(s_t, a_t)$. The agent's objective is to find a policy π that maximizes the expected sum of discounted future rewards

$$J^\pi := \mathbb{E}_{\tau \sim P^\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{10}$$

where $\tau = (s_0, a_0, r_0, s_1, ...)$ represents a trajectory through the environment and P^π is the distribution over trajectories induced by following policy π. An optimal policy $\pi^* \in \Pi^* := \arg\max_\pi J^\pi$ can be found through the optimal value function $V^* : \mathcal{S} \to \mathbb{R}$ or the optimal action-value function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which satisfy the Bellman optimality equations [9], [10]

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a), \tag{11}$$

$$Q^*(s, a) = \mathbb{E}_{r, s' \sim P_{R,S}(s,a)}\left[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')\right]. \tag{12}$$

This framework serves as the building block for more sophisticated models like Contextual MDPs and Bayesian approaches to reinforcement learning, as described in the following sections.

2.4 Contextual Reinforcement Learning

In contextual RL, we use the concept of a Contextual MDP.

> **Contextual MDP:** A Markov Decision Process augmented with a context variable that determines the specific dynamics of the environment [11]. This allows us to model uncertainty about the true environment through uncertainty about the context.

In a Contextual Markov Decision Process (CMDP), we work in an infinite-horizon, discounted setting where a context variable $\varphi \in \Phi \subseteq \mathbb{R}^d$ indexes specific MDPs. Formally, we describe this as

$$\mathcal{M}(\varphi) := \langle \mathcal{S}, \mathcal{A}, P_0, P_S(s, a, \varphi), P_R(s, a, \varphi), \gamma \rangle. \tag{13}$$

where the context φ parametrizes both:

- A transition distribution $P_S(s, a, \varphi)$ determining how states evolve;
- A reward distribution $P_R(s, a, \varphi)$ determining the rewards received.

The agent has complete knowledge of the following aspects of the environment:

- The state space $\mathcal{S} \subset \mathbb{R}^n$;
- The action space $\mathcal{A}$;
- The initial state distribution P_0;
- The discount factor γ.

However, the agent does not know the true context φ^* that determines the actual dynamics and rewards.

2.4.1 Policies and Histories

In contextual RL, an agent follows a *context-conditioned policy* $\pi : \mathcal{S} \times \Phi \to \mathcal{P}(\mathcal{A})$, selecting actions according to $a_t \sim \pi(s_t, \varphi)$. As the agent interacts with the environment, it accumulates a history of experiences $h_t := \{s_0, a_0, r_0, s_1, a_1, r_1, ..., a_{t-1}, r_{t-1}, s_t\}$. This history belongs to a state-action-reward product space $\mathcal{H}_t$ and follows a context-conditioned distribution $P_t^\pi(\varphi)$ with density

$$p_t^\pi(h_t|\varphi) = p_0(s_0) \prod_{i=0}^{t} \pi(a_i|s_i, \varphi) p(r_i, s_{i+1}|s_i, a_i, \varphi). \tag{14}$$

2.4.2 Optimization Objective

The agent's goal in a CMDP is to find a policy that optimizes the expected discounted return

$$J^\pi(\varphi) = \mathbb{E}_{\tau_\infty \sim P_\infty^\pi(\varphi)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{15}$$

An optimal policy $\pi^*(\cdot, \varphi)$ belongs to the set $\Pi_\Phi^*(\varphi) := \arg\max_{\pi \in \Pi_\Phi} J^\pi(\varphi)$. With this, we define the optimal Q-function $Q^*(h_t, a_t, \varphi)$.

> **Optimal Q-Function:** For an optimal policy π^*, *the optimal Q-function* $Q^* : \mathcal{S} \times \mathcal{A} \times \Phi \to \mathbb{R}$ satisfies the Bellman equation
>
> $$\mathcal{B}^*[Q^*](s_t, a_t, \varphi) = Q^*(s_t, a_t, \varphi), \tag{16}$$
>
> where $\mathcal{B}^*$ is the optimal Bellman operator defined as
>
> $$\mathcal{B}^*[Q^*](s_t, a_t, \varphi) := \mathbb{E}_{r_t, s_{t+1} \sim P_{R,S}(s_t, a_t, \varphi)} \left[r_t + \max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a', \varphi) \right]. \tag{17}$$

2.4.3 The Learning Challenge

When an agent has access to the true MDP $\mathcal{M}(\varphi^*)$, finding an optimal policy becomes a *planning problem*. However, in real-world scenarios, agents typically lack access to the true transition dynamics and reward functions. This transforms the task into a *learning problem*, where the agent must balance:

1. *Exploration*: learning about the environment's dynamics through interaction;
2. *Exploitation*: using current knowledge to maximize rewards.

This tension — known as the exploration/exploitation dilemma — remains one of the core challenges in reinforcement learning. As we'll see in the next section, Bayesian approaches offer a principled framework for addressing this challenge.

2.5 Bayesian Reinforcement Learning

In the Bayesian approach to RL, rather than viewing uncertainty as a problem to be eliminated, it becomes an integral part of the decision-making process — something to be reasoned about systematically [12].

> **Bayesian Epistemology:** A framework that characterizes uncertainty through probability distributions over possible worlds. In reinforcement learning, this means maintaining distributions over possible MDPs, updated as new evidence arrives [12].

2.5.1 From Prior to Posterior

The Bayesian learning process starts with a *prior distribution* P_Φ representing our initial beliefs about the true context φ^* before any observations. As the agent interacts with the environment, it accumulates a history of experiences h_t and updates these beliefs through Bayesian inference, forming a *posterior distribution* $P_\Phi(h_t)$.

This history-dependent posterior in Bayesian RL differentiates it from traditional RL approaches.

> **History-Conditioned Policies:** Unlike traditional RL policies that map states to actions, Bayesian policies operate on entire histories, defining a set of history-conditioned policies $\Pi_{\mathcal{H}} := \{\pi : \mathcal{H} \to \mathcal{P}(\mathcal{A})\}$, where $\mathcal{H} := \{\mathcal{H}_t | t \geq 0\}$ denotes the set of all histories [12].

Where the prior P_Φ represents our initial uncertainty (the special case where $h_t = \emptyset$), the posterior $P_\Phi(h_t)$ captures our refined beliefs after observing interactions with the environment. This allows us to reason about future outcomes by marginalizing across all possible MDPs according to our current uncertainty.

2.5.2 The Bayesian Perspective on Transitions

The power of the Bayesian approach stems from how it handles state transitions. Instead of committing to a single model of the environment, it maintains a distribution over possible transitions through the *Bayesian state-reward transition distribution*

$$P_{R,S}(h_t, a_t) := \mathbb{E}_{\varphi \sim P_\Phi(h_t)}\left[P_{R,S}(s_t, a_t, \varphi)\right]. \tag{18}$$

This distribution lets us reason about future trajectories using the *prior predictive distribution* P_t^π with density

$$p_t^\pi(h_t) = p_0(s_0) \prod_{i=0}^{t} \pi(a_i|h_i) p(r_i, s_{i+1}|h_i, a_i). \tag{19}$$

The belief transition distribution $P_{\mathcal{H}}(h_t, a_t)$ captures how our beliefs evolve with new observations, with density

$$p_{\mathcal{H}}(h_{t+1}|h_t, a_t) = p(s_{t+1}, r_t|h_t, a_t). \tag{20}$$

This formulation leads to the definition of the Bayes-adaptive MDP (BAMDP) [13]

$$\mathcal{M}_{\text{BAMDP}} := \langle \mathcal{H}, \mathcal{A}, P_0, P_{\mathcal{H}}(h,a), \gamma \rangle. \tag{21}$$

2.5.3 Natural Resolution of the Exploration Dilemma

An interesting aspects of the Bayesian framework is how it naturally resolves the exploration-exploitation dilemma [12], [13]. Rather than treating exploration as a separate mechanism, it emerges naturally from the optimization of expected returns under uncertainty

$$J^{\pi}_{\text{Bayes}} := \mathbb{E}_{h_\infty \sim P^{\pi}_{\infty}} \left[\sum_{i=0}^{\infty} \gamma^i r_i \right]. \tag{22}$$

A Bayes-optimal policy achieves perfect balance between exploration and exploitation because:

1. It accounts for uncertainty through the posterior at each timestep;
2. It considers how this uncertainty will evolve in the future;
3. It weights future information gain by the discount factor γ.

2.5.4 The Optimal Bayesian Q-Function

For a Bayes-optimal policy π^*, we can define the optimal Bayesian Q-function as $Q^*(h_t, a_t) := Q^{\pi^*_{\text{Bayes}}}(h_t, a_t)$. This Q-function satisfies the optimal Bayesian Bellman equation

$$Q^*(h_t, a_t) = \mathcal{B}^*[Q^*](h_t, a_t), \tag{23}$$

where $\mathcal{B}^*[Q^*]$ is the optimal Bayesian Bellman operator

$$\mathcal{B}^*[Q^*](h_t, a_t) := \mathbb{E}_{h_{t+1} \sim P_{\mathcal{H}}(h_t, a_t)} \left[r_t + \gamma \max_{a'} Q^*(h_{t+1}, a') \right]. \tag{24}$$

2.6 Bayesian Exploration Networks

Model-free reinforcement learning takes a different approach to learning optimal behaviors compared to model-based methods. Rather than explicitly modeling the environment's dynamics, model-free approaches attempt to learn optimal policies from experience. Bayesian Exploration Networks (BENs) extend this idea into the Bayesian realm by characterizing uncertainty in the Bellman operator itself, instead of in the environment's transition dynamics [7].

> **Model-Free vs Model-Based:** While model-based approaches maintain explicit probabilistic models of the environment's dynamics, model-free methods like BEN directly learn mappings from states to values or actions [14]. This can be more computationally efficient but requires careful handling of uncertainty.

2.6.1 The Bootstrapping Perspective

Instead of modeling the full complexity of state transitions, we can use bootstrapping to estimate the optimal Bayesian Bellman operator directly. Given samples from the true reward-state distribution $r_t, s_{t+1} \sim P^*_{R,S}(s_t, a_t)$, we estimate $b_t = \beta_\omega(h_{t+1}) \coloneqq r_t + \gamma \max_{a'} Q_\omega(h_{t+1}, a')$.

This bootstrapping process can be viewed as a transformation of variables — mapping from the space of rewards and next states to a single scalar value. This significantly reduces the dimensionality of the problem while preserving the essential information needed for learning optimal policies [7].

> **Bootstrapped Distribution:** The samples b_t follow what we call the Bellman distribution $P^*_B(h_t, a_t; \omega)$, which captures the distribution of possible Q-value updates [7]. This distribution encapsulates both the environment's inherent randomness and our uncertainty about its true nature.

2.6.2 Sources of Uncertainty

When predicting future Q-values, BEN distinguishes between two types of uncertainty.

1. **Aleatoric Uncertainty**: The inherent randomness in the environment's dynamics that persists even with perfect knowledge.

   Example: Rolling a fair die — this uncertainty cannot be reduced with more data.

2. **Epistemic Uncertainty**: Our uncertainty about the true Bellman distribution itself. This represents our lack of knowledge about the environment and can be reduced through exploration and learning.

   Example: Determining whether a die is fair — this uncertainty is can be reduced with more data.

This separation of uncertainties allows BEN to distinguish between what is fundamentally unpredictable (aleatoric) and what can be learned through exploration (epistemic), leading to more efficient learning strategies [7].

2.6.3 Network Architecture

BEN implements this uncertainty handling through three neural networks [7]:

1. **Recurrent Q-Network**:

   At its core, BEN uses a recurrent neural network (RNN) to approximate the optimal Bayesian Q-function. The Q-network processes the entire history of interactions. We denote the output at timestep t as $q_t = Q_\omega(h_t, a_t) = Q_\omega\left(\hat{h}_{t-1}, o_t\right)$, where h_t represents the history up to time t, a_t is the action, $\hat{h}_{t-1}$ is the recurrent encoding of previous history, and o_t contains the current observation tuple $\{r_{t-1}, s_t, a_t\}$. By conditioning on history rather than just current state, BENs can capture how uncertainty evolves over time, making it capable of learning Bayes-optimal policies [7].

2. **Aleatoric Network**:

The aleatoric network models the inherent randomness in the environment. It uses normalizing flows to transform a simple base distribution (such as a standard Gaussian) into a more complex distribution $P_B(h_t, a_t, \varphi; \omega)$ over possible next-state Q-values, representing the aleatoric uncertainty in the Bellman operator, by applying the transformation $b_t = B(z_{\text{al}}, q_t, \varphi)$ [7], where

- $z_{\text{al}} \in \mathbb{R} \sim P_{\text{al}}$ is a base variable with a zero-mean, unit variance Gaussian P_{al};
- q_t is the Q-value from the recurrent network;
- and φ and ω represent the network parameters.

We optimize the parameters ω of the recurrent Q-network and the aleatoric network using the Mean Squared Bayesian Bellman Error (MSBBE), which satisfies the optimal Bayesian Bellman equation for our Q-function approximator.

3. **Epistemic Network**:

   The epistemic network captures our uncertainty about the environment itself. The network maintains a dataset of bootstrapped samples $\mathcal{D}_\omega(h_t) := \{(b_i, h_i, a_i)\}_{i=0}^{t-1}$ collected from interactions with the environment. Each tuple in this dataset consists of

   - a bootstrapped value estimate b_i;
   - the history at that timestep h_i;
   - the action taken a_i.

   Given this dataset, we would ideally compute the posterior distribution $P_\Phi(\mathcal{D}_\omega(h_t))$ representing our refined beliefs about the environment after observing these samples. However, computing this posterior directly is typically intractable for complex environments [7]. Instead, BEN employs normalizing flows for variational inference.

   The epistemic network learns a tractable approximation P_ψ parametrized by $\psi \in \Psi$ that aims to capture the essential characteristics of the true posterior. We optimize this approximation by minimizing the KL-divergence between our approximation and the true posterior:

$$\text{KL}\big(P_\psi \parallel P_\Phi(\mathcal{D}_\omega(h_t))\big). \tag{25}$$

   This optimization is performed indirectly by maximizing the Evidence Lower Bound (ELBO) $\text{ELBO}(\psi; h, \omega)$, which is equivalent as proved by [7].

2.6.4 Training Process

The network is trained through a dual optimization process:

1. **MSBBE Optimization:** The Mean Squared Bayesian Bellman Error (MSBBE) is computed as the difference between the predictive optimal Bellman operator $B^+[Q_\omega]$ and Q_ω [7]:

$$\text{MSBBE}(\omega; h_t, \psi) := \|B^+[Q_\omega](h_t, a_t) - Q_\omega(h_t, a_t)\|_\rho^2, \tag{26}$$

   which is minimized to learn the parametrisation ω^*, satisfying the optimal Bayesian Bellman equation for our Q-function approximator, with ρ being an arbitrary sampling distribution with support over $\mathcal{A}$.

The predictive optimal Bellman operator can be obtained by taking expectations over variable b_t using the predictive optimal Bellman distribution $P_B(h_t, a_t; \omega)$:

$$B^+[Q_\omega](h_t, a_t) := \mathbb{E}_{b_t \sim P_B(h_t, a_t; \omega)}[b_t], \tag{27}$$

where $P_B(h_t, a_t; \omega) = \mathbb{E}_{\varphi \sim P_\Phi(\mathcal{D}_\omega(h_t))}[P_B(h_t, a_t, \varphi; \omega)]$.

This gives rise to a nested optimisation problem, as is common in model-free RL [7], which can be solved using two-timescale stochastic approximation [15]. In the case of BEN, we update the epistemic network parameters ψ using gradient descent on an asymptotically faster timescale than the function approximator parameters ω to ensure convergence to a fixed point, as proposed by [7].

2. **ELBO Optimization:** The Evidence Lower BOund (ELBO) serves as the optimization objective for training BEN's epistemic network. While minimizing the KL-divergence $\mathrm{KL}\big(P_\psi \parallel P_\Phi(\mathcal{D}_\omega(h_t))\big)$ directly would give us the most accurate approximation of the true posterior, computing this divergence is typically intractable. Instead, we can derive and optimize the ELBO, which provides a tractable lower bound on the model evidence [7].

   By applying Baye's rule on this KL-divergence, [7] derives

$$\begin{aligned} &\mathrm{ELBO}(\psi; h_t, \omega) \\ &\quad := \mathbb{E}_{z_{\mathrm{ep}} \sim P_{\mathrm{ep}}}\left[\sum_{i=0}^{t-1}\Big(B^{-1}(b_i, q_i, \varphi)^2 - \log|\ \partial_b B^{-1}(b_i, q_i, \varphi)\ |\Big) - \log p_\Phi(\varphi)\right], \end{aligned} \tag{28}$$

   where $\varphi = t_\psi\big(z_{\mathrm{ep}}\big)$ and:

   - z_{ep} is drawn from the base distribution P_{ep} (a standard Gaussian $\mathcal{N}\big(0, I^d\big)$);
   - B^{-1} is the inverse of the aleatoric network's transformation;
   - $\partial_b B^{-1}$ is the Jacobian of this inverse transformation;
   - t_ψ represents the epistemic network's transformation.

   > **Jacobian Term:** The term $\partial_b B^{-1}$ accounts for how the epistemic network's transformation changes the volume of probability space. This is important for maintaining proper probability distributions when using normalizing flows [16].

   The ELBO objective breaks down into three key components:

   1. A reconstruction term $B^{-1}(b_i, q_i, \varphi)^2$ that measures how well our model can explain the observed Q-values;
   2. A volume correction term $\log|\partial_b B^{-1}(b_i, q_i, \varphi)|$ that accounts for the change in probability space;
   3. A prior regularization term $\log p_\Phi(\varphi)$ that encourages the approximated posterior to stay close to our prior beliefs.

By minimizing the ELBO, we obtain an approximate posterior that balances accuracy with computational tractability, allowing BEN to maintain and update its uncertainty estimates efficiently during learning [7].

> **Training Dynamics:** The two optimization processes occur at different timescales, with epistemic updates happening more frequently than the Q-network updates. This separation ensures stable convergence while maintaining the ability to adapt to new information [15].

With this architecture, BEN can learn truly Bayes-optimal policies while maintaining the computational efficiency of model-free methods [7]. This makes it particularly well-suited for environments with sparse, delayed rewards where efficient exploration is important.

3 Theoretical Framework

In this section, we develop a theoretical framework for comparing GFlowNets and Bayesian Exploration Networks (BENs) in environments with delayed and sparse rewards. Our goal is to establish precise criteria for evaluating these different approaches to exploration and uncertainty handling.

3.1 Problem Formulation

Consider an environment with delayed rewards characterized by

- **Reward Delay:** The temporal gap T_{reward} between an action and its corresponding reward signal . We formally define this as

$$T_{\text{reward}} := \min\{t \mid s_t \in \mathcal{X}, r_t \neq 0\}. \tag{29}$$

  In our n-chain environment, T_{reward} corresponds to the chain length.

- **Reward Sparsity:** The proportion ρ of state-action pairs that yield non-zero rewards:

$$\rho := \frac{|\{(s,a) \in \mathcal{S} \times \mathcal{A} : \mathbb{E}[R(s,a)] \neq 0\}|}{|\mathcal{S} \times \mathcal{A}|}, \tag{30}$$

  where $R(\cdot)$ is some reward distribution.

These characteristics create distinct challenges, as discussed, for reinforcement learning algorithms.

1. The *temporal credit assignment problem* becomes more severe with increasing T_{reward}.
2. The *exploration efficiency* becomes critical as ρ decreases.
3. The *signal-to-noise ratio* in value estimation deteriorates with both T_{reward} and ρ.

3.1.1 Value Propagation Mechanisms

The algorithms differ in how they handle value propagation. GFlowNet value propagation maintains consistency between forward and backward flows through the trajectory balance constraint $Z \prod_{t=1}^{n} P_F(s_t|s_{t-1}) = F(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t)$.

BEN value propagation directly models the distribution of bootstrapped values through the estimated Bellman operator $b_t = r_t + \gamma \max_{a'} Q_\omega(h_{t+1}, a')$.

3.1.2 Uncertainty Representation

Both approaches maintain uncertainty estimates but through different mechanisms:

- GFlowNets implicitly capture uncertainty through the learned flow distribution;
- BENs explicitly separate aleatoric and epistemic uncertainty.

This leads to our central hypothesis.

> **Hypothesis:** In environments with highly delayed rewards (large T_{reward}), BEN's explicit uncertainty decomposition leads to more efficient learning compared to GFlowNets, particularly in early training stages.

However, this advantage diminishes as T_{reward} decreases. This hypothesis is supported by the following three observations.

1. BEN's direct modeling of the Bellman operator allows for faster value propagation.
2. The explicit separation of uncertainty types enables more targeted exploration.
3. GFlowNets must learn complete trajectories before gaining signal about reward structure.

3.1.3 Analytical Framework

To evaluate our hypothesis about the relative performance of GFlowNets and BENs in delayed reward environments, we establish three metrics that capture different aspects of learning and exploration efficiency.

1. **Sample Efficiency:** Measures how quickly each algorithm converges to optimal behavior through their respective loss functions.

   For GFlowNets, we track the trajectory balance loss $\mathcal{L}_{\text{TB}}(\tau) = \left(\log Z_\theta + \log \sum_{t=1}^{n} P_F(s_t|s_{t-1};\theta) - \log R(x) - \log \sum_{t=1}^{n} P_B(s_{t-1}|s_t;\theta)\right)^2$.

   While for BEN, we monitor the Mean Squared Bayesian Bellman Error $\text{MSBBE}(\omega; h_t, \psi) = \|B^+[Q_\omega](h_t, a_t) - Q_\omega(h_t, a_t)\|_\rho^2$.

   These metrics allow us to quantify learning progress and compare convergence rates between algorithms as a function of reward delay T_{reward}.

2. **Distribution Matching:** Evaluates how well the learned policy matches the true underlying reward structure.

   In our n-chain environment, where terminal states are guaranteed to be reached, we measure the KL-divergence between the true terminal state distribution P (determined by rewards) and the empirical distribution Q generated by each algorithm $\text{KL}(P \parallel Q)$. This metric is particularly relevant for GFlowNets, as they explicitly aim to learn a sampling distribution proportional to the reward function.

3. **Exploration Efficiency:** Captures how effectively each algorithm explores the state space before converging to optimal behavior.

   We introduce two complementary metrics.

   - **State Coverage Ratio:** Measures the proportion of the state space explored over time as

   $$E(t) := \frac{|S_{\text{visited}}(t)|}{|S|}, \tag{31}$$

   where $S_{\text{visited}}(t)$ represents the set of unique states visited up to time t.

   - **Time-to-First-Success:** Quantifies initial exploration effectiveness as

   $$T_{\text{success}} := \min\{t \mid s_t \in \mathcal{X}, r_t > 0\}. \tag{32}$$

   This metric becomes increasingly important as reward delay T_{reward} grows, as it indicates how quickly each algorithm can discover successful trajectories in sparse reward settings.

Combining these metrics, we construct a evaluation framework that addresses three important aspects of performance:

1. Learning efficiency through loss convergence analysis;
2. Policy quality through distribution matching;
3. Exploration effectiveness through coverage and discovery time.

This framework allows us to investigate how the advantage of BEN's explicit uncertainty decomposition versus GFlowNet's flow-based approach vary with reward delay T_{reward} and sparsity ρ. In particular, we can test our hypothesis that BEN's advantages become more pronounced as T_{reward} increases by examining the correlation between reward delay and relative performance across the mentioned metrics.

4 Experimental Design

We implement a modified n-chain environment that serves as a testbed for studying delayed rewards. This environment presents properties that make it particularly suitable for our analysis.

> **N-Chain Environment:** A sequential decision-making environment with a branching structure, where rewards are only received at terminal states.

Parameters of the n-chain environment include a chain length n, controlling reward delay T_{reward}, a branching factor b, affecting exploration complexity, and terminal state rewards, determining optimal distributions. Adjusting the chain length n and branching factor b also allow us to control the reward sparsity ρ by proxy, as a longer chain involves more states, increasing sparsity. Similarly, increasing the branching factor b introduces more branches, again inreasing the number of states and, thus, the sparsity ρ as well.

The environment consists of three main components:

1. **State Space:** A chain of length n with a branching point at the middle ($\lfloor \frac{n}{2} \rfloor$), creating multiple possible trajectories. This results in a total of $\lfloor \frac{n}{2} \rfloor + b(n - \lfloor \frac{n}{2} \rfloor)$ possible states, and exactly b terminal states.
2. **Action Space:** At each state, an agent can move forward with the FORWARD action, or stay in terminal states with the TERMINAL_STAY action. At the split point, the agent must choose a branch using the BRANCH_i action, where $i \in \{1, ..., b\}$. This results in a total of $2 + b$ actions.
3. **Reward Structure:** Generally, a reward function $R : \mathcal{X} \to \mathbb{R}$ is defined over terminal states $x \in \mathcal{X}$, creating a natural target distribution for sampling and clear optimal policies. For GFlowNets, the reward function is further constrained to the domain $\mathbb{R}_{>0}$, yielding a reward function $R : \mathcal{X} \to \mathbb{R}_{>0}$.

This design creates a sparse reward landscape — agents must execute sequences of $\lfloor \frac{n}{2} \rfloor - 1$ actions before reaching the branch point, where the chosen branch determines the final reward, followed by another $\lfloor \frac{n}{2} \rfloor$ actions to reach any terminal state. This structure allows us to precisely control both reward delay T_{reward} and sparsity ρ.

4.1 Evaluation Protocol

We evaluate each algorithm through a sequence of experiments with increasing complexity.

1. **Base Configuration:**
   - Fixed chain length ($n = 5$);
   - Three terminal states with fixed rewards $\{10, 20, 70\}$;
   - BEN: Discount factor $\gamma = 0.9$;
   - GFlowNet: Exploration factor $\varepsilon = 0.1$.
2. **Delay Variation Studies:**
   - Chain lengths $n \in \{3, 5, 7, 9, 11\}$;
   - Keeping terminal rewards fixed;
   - Measuring performance vs. delay T_{reward}.
3. **Stochastic Analysis:**
   - Introducing random rewards drawn from distributions;
   - Terminal states $x \in \mathcal{X}$ rewards: $R_x \sim \mathcal{N}(\mu_x, \sigma^2)$;
   - Testing robustness to uncertainty.

For each configuration, we conduct 10 independent trials with different random seeds to minimize the impact of statistical variance. We then apply the framework discussed in Section 3.1.3 on the results of these three configurations for analysis.

4.2 Implementation Details

4.2.1 Environment Setup

Our implementation of the n-chain environment creates a decision space that enables precise control over reward delay and sparsity. The environment is implemented as a deterministic MDP,

where each state is encoded as a composite tensor of shape `[n*3 + 4]`, consisting of a one-hot position encoding of length 3n to account for all three branches, as well as a one-hot branch encoding of length 4 (pre-split + 3 possible branches).

The state space is managed through a `NChainState` class that tracks three attributes:

1. *Position:* An integer in [0, n-1] indicating location in the chain;
2. *Branch:* An integer flag (-1 for pre-split, {0,1,2} for branch selection);
3. *Chain Length:* The parameter n that determines the delay between actions and rewards.

The action space is managed through a `NChainAction` enum and consists of the following five distinct actions:

- `TERMINAL_STAY`: Available only in terminal states;
- `FORWARD`: For progression along the chosen path;
- `BRANCH_0`, `BRANCH_1`, `BRANCH_2`: Available only at the split point.

The environment enforces strict action masking through a get_valid_actions method that returns only legitimate actions for each state. This creates three distinct decision phases:

1. Pre-split: Only `FORWARD` actions are valid;
2. Split-point: Only `BRANCH_i` actions are valid, for $i \in \{0, 1, 2\}$;
3. Post-split: `FORWARD` until terminal, then `TERMINAL_STAY`.

For interaction with the environment, the implementation provides methods like `step`, `reset`, as well as utility methods like state-to-tensor conversions. This approach allows for systematic variation of both reward delay through the chain length n, and reward sparsity through the ratio of rewarding states to total states.

4.2.2 Network Architectures

The GFlowNet implementation consists of three primary components working in concert to learn flow-matching policies:

1. *State Encoder:* A multi-layer perceptron that processes state tensors of shape `[batch_size, state_dim]` into a learned encoding of shape `[batch_size, hidden_dim]`. This encoding captures the essential features of each state necessary for policy decisions.
2. *Forward Policy Network:* A single dense layer that transforms the state encoding into a forward policy distribution over actions, outputting tensors of shape `[batch_size, num_actions]`. This network determines the probabilities of taking each possible action from the current state.
3. *Backward Policy Network:* For the n-chain environment, this component is simplified due to the tree structure of the state space. Since each non-initial state has exactly one parent, the backward policy becomes deterministic, requiring only a lightweight network layer to maintain architectural symmetry.

Additionally, we maintain a scalar parameter representing $\log Z$ (the partition function), which is important for our flow matching approach and is learned alongside the network parameters.

The BEN implementation is heavily based on the implementation provided by [7], but is adapted for the n-chain environment. It comprises three interacting networks:

1. *Q-Bayes Network:* A recurrent neural network that processes observation tuples (state, action, reward) to generate Q-values and maintain a history encoding. The network accepts inputs of shape `[batch_size, state_dim + action_dim + reward_dim]` and outputs both Q-values `[batch_size, num_actions]` and a RNN hidden state `[batch_size, rnn_hidden_dim]` for further processing.
2. *Aleatoric Network:* Implements a normalizing flow to model the inherent randomness in Bellman updates through:
   - A conditioning network (ConditionerMLP) [7] that generates parameters for the flow based on Q-values and RNN state;
   - An inverse autoregressive flow that transforms a base distribution into the desired Bellman distribution.
3. *Epistemic Network:* Another normalizing flow that captures uncertainty about the environment itself, transforming a base variable $z_{\text{ep}} \in \mathbb{R}^d$ into the parameter space that defines our beliefs about the environment.

4.2.3 Training Procedures

Training of the GFlowNet follows a tempered exploration strategy where:

1. The forward policy is "softened" during training using an ε-greedy approach with $\varepsilon = 0.1$, allowing for off-policy exploration while maintaining flow-matching properties:

   ```
   (1 - self.epsilon) * policy + self.epsilon * random_policy.
   ```

2. The trajectory balance loss is minimized using stochastic gradient descent with the Adam optimizer [17]:

   ```
   L_TB(tau) = (log_Z + sum_log_pf - log_R - sum_log_pb).pow(2),
   ```

   where `tau` represents a trajectory, `sum_log_pf` represents the sum of log forward probabilities over `tau`, and `sum_log_pb` similarly represents the sum of log backward probabilities over `tau`.

BEN employs a two-timescale optimization process:

1. *Fast Timescale:* Updates to the epistemic network parameters ψ through ELBO minimization:

   ```
   ELBO(psi; h, omega) = -log_p - torch.mean(log_q) - 1 / (time_period + 1) * prior,
   ```

   where `log_p` represents the base variable z_{al} obtained from $B^{-1}(b_i, q_i, \varphi)^2$, `torch.mean(log_q)` represents the log Jacobian of this base variable $\log|\partial_b B^{-1}(b_i, q_i, \varphi)|$, and `1 / (time_period + 1) * prior` represents the log prior $\log p_\Phi(\varphi)$.

2. *Slow Timescale:* Updates to the Q-network parameters ω through MSBBE minimization:

   ```
   MSBBE(omega; h_t, psi) = torch.abs((b1 - q) * (b2 - q)),
   ```

where `b1` and `b2` represent two samples from the predictive Bellman operator $B^+[Q_\omega](h_t, a_t)$, and `q` represents the Q-value obtained from $Q_\omega(h_t, a_t)$. This operation is similar to a simple squared error, with the only difference being the use of two different samples, minimizing statistical variance.

This separation of timescales ensures stable convergence while maintaining the ability to adapt to new information, controlled by the discount factor γ. For details about hyperparameter selection, we refer to Appendix A.1.

5 Results and Analysis

NOTE

- Quantitative results
  - Performance comparisons
  - Statistical analysis
- Qualitative analysis
  - Exploration patterns
  - Learning behavior
- Discussion of findings

5.1 Convergence Analysis

TODO

- Loss curves across different chain lengths
  - Show plot for n=3, n=7, and n=11
  - Rest in appendix
- Statistical significance tests
  - T-tests between early/mid/late stages
- Convergence time analysis
  - By what iteration does loss seem to converge?

5.2 Exploration Efficiency

TODO

- State visitation heat maps
- First-success timing metrics
- Coverage rate comparisons

5.3 Policy Quality Assessment

TODO

- KL divergence from optimal policy
- Terminal state distribution analysis
- Reward accumulation curves

5.4 Learning Dynamics

TODO

- Early exploration strategies
- Transition points in policy development
- Adaptation to different reward scales

5.5 Failure Mode Analysis

TODO

- Common pitfalls in learning
- Edge case behaviors
- Stability considerations

6 Conclusion

NOTE

- Summary of contributions
- Key insights
- Future work directions

6.1 Future Research

NOTE

Everything I think I could have done better essentially.

Bibliography

[1] R. S. Sutton and Barto, *Reinforcement learning : an introduction /*, Second edition. in Adaptive computation and machine learning. Cambridge, Massachusetts ;: The MIT Press, 2020.

[2] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "VIME: Variational Information Maximizing Exploration." [Online]. Available: https://arxiv.org/abs/1605.09674

[3] A. Harutyunyan *et al.*, "Hindsight Credit Assignment." [Online]. Available: https://arxiv.org/abs/1912.02503

[4] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep Exploration via Bootstrapped DQN." [Online]. Available: https://arxiv.org/abs/1602.04621

[5] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction." [Online]. Available: https://arxiv.org/abs/1705.05363

[6] E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio, "Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation." [Online]. Available: https://arxiv.org/abs/2106.04399

[7] M. Fellows, B. Kaplowitz, C. S. de Witt, and S. Whiteson, "Bayesian Exploration Networks." [Online]. Available: https://arxiv.org/abs/2308.13049

[8] N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio, "Trajectory balance: Improved credit assignment in GFlowNets." [Online]. Available: https://arxiv.org/abs/2201.13259

[9] M. L. Puterman, *Markov decision processes : Discrete stochastic dynamic programming*. in Wiley series in probability and mathematical statistics. Applied probability and statistics section. New York: Wiley, 1994.

[10] R. Bellman, *Dynamic programming /*. in A Rand corporation research study. Princeton,N.J: Princeton univer.press, 1962.

[11] A. Hallak, D. D. Castro, and S. Mannor, "Contextual Markov Decision Processes." [Online]. Available: https://arxiv.org/abs/1502.02259

[12] M. Ghavamzadeh, S. Mannor, J. Pineau, and A. Tamar, "Convex Optimization: Algorithms and Complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 5–6, pp. 359–483, 2015, doi: 10.1561/2200000049.

[13] M. O. Duff and A. G. Barto, "Optimal learning: computational procedures for bayes-adaptive markov decision processes," 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:118819159

[14] P. Dayan and Y. Niv, "Reinforcement learning: The Good, The Bad and The Ugly," *Current opinion in neurobiology*, vol. 18, no. 2, pp. 185–196, 2008.

[15] V. S. Borkar, *Stochastic Approximation : A Dynamical Systems Viewpoint*, 1st ed. 2008. in Texts and Readings in Mathematics ; 48. Gurgaon: Hindustan Book Agency, 2008.

[16] D. Rezende and S. Mohamed, "Variational Inference with Normalizing Flows," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., in Proceedings of Machine Learning Research, vol. 37. Lille, France: PMLR, 2015, pp. 1530–1538. [Online]. Available: https://proceedings.mlr.press/v37/rezende15.html

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." [Online]. Available: https://arxiv.org/abs/1412.6980

Appendix

A Implementation Details

A.1 Hyperparameter selection

GFlowNet Hyperparameters:

- Hidden dimension: 64;
- Learning rate: 1e-4;
- Exploration ε: 0.1;
- Batch size: 32.

BEN Hyperparameters:

- RNN hidden dimension: 64;
- Learning rate (Q-network): 1e-4;
- Learning rate (Epistemic network): 1e-4;
- Base dimension (z_{ep}): 8;
- Discount factor γ: 0.9,
- Batch size: 32.