# A GRID WORLD AGENT WITH FAVORABLE INDUCTIVE BIASES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

We present a novel experiential learning agent with causally-informed intrinsic reward that is capable of learning sequential and causal dependencies in a robust and data-efficient way within grid world environments. After reflecting on state-of-the-art Deep Reinforcement Learning algorithms, we provide a relevant discussion of common techniques as well as our own systematic comparison within multiple grid world environments. Additionally, we investigate the conditions and mechanisms leading to data-efficient learning and analyze relevant inductive biases that our agent utilizes to effectively learn causal knowledge and to plan for rewarding future states of greatest expected return.

## 1 INTRODUCTION

Grid world environments come in many forms and have been studied extensively in the history of Artificial Intelligence with some notable examples such as Wumpus World (Bryce, 2011), Minigrid (Chevalier-Boisvert et al., 2024), and Tileworld (Pollack & Ringuette, 1990). However, the creation of intelligent grid world agents capable of learning effectively and in a data-efficient way has posed significant challenges. Current Reinforcement Learning (RL) agents struggle in some instances due to sequential dependencies, partial observability (Wang et al., 2023a), continual learning (primacy bias Kim et al. (2024), stability-plasticity dilemma Anand & Precup (2024)), relatively high-dimensional state spaces, compared to more traditional RL tasks (even when a single value per cell is provided rather than per pixel), and sometimes non-deterministic effects of actions.

Sequential dependencies usually raise the training data demand exponentially depending on the combinatorics, and ultimately the number of the arising options, unless the agent is capable of learning causal representations that transfer well because chaining them is favorable for reaching an intended outcome. Partial observability, on the other hand, may require the model to have access to information from prior states, which typically corresponds to the previously observed values in a grid world outside the agent's current field of view. In terms of input dimensionality, there is a trade-off between observation window being too small for learning an effective policy and the agent's observation window being more high-dimensional thereby demanding more training data, even though this might be the least problematic. Additionally, if the agent is able to represent values outside of its observation window, a learned policy needs to consider not only the observation window itself but also how it spatially relates to the remembered information beyond it.

With these considerations in mind, we introduce Non-Axiomatic Causal Explorer (NACE), a novel experiential learning agent, which leverages causal reasoning and intrinsic reward signals to enable more efficient learning as well as possesses learning mechanisms with the involved inductive biases. NACE is designed to induce causal rules from temporal and spatial local changes in the grid, which are often (but not always) caused by the agent. It utilizes these rules to plan for and reach future states of maximum uncertainty in order to effectively learn more (causal rules) about the environment, thereby improving on predictability-based intrinsic reward formulations.

To illustrate the effectiveness of our approach, we provide a comprehensive discussion of state-of-the-art Deep Reinforcement Learning (DRL) techniques as well as our own systematic comparison within multiple grid world environments demonstrating our agent's remarkable improvement in data efficiency, achieving similar performance with about 1000 samples where deep-RL algorithms typically require 1 million samples, representing a 1000-fold reduction in data requirements. We also

thoroughly investigate the conditions and mechanisms under which learning in grid world environments can become more data-efficient and attempt to answer the question about which inductive biases can lead to close-to-optimal learning speeds in the case where the agent is not pre-equipped with particular interaction rules between grid cell types but has the inductive biases to build and use them effectively. Lastly, we analyze useful inductive biases applicable among a wide range of grid worlds and their generality to raise data efficiency of learning in such domains.

## 2 Related Work

Current RL techniques such as value-based (Deep Q-Learning (Mnih et al., 2013)) and policy gradient-based (Proximal Policy Optimization (Schulman et al., 2017)) typically require millions of training iterations to solve grid-world environments (Zhang et al., 2020b), struggling to capture causal dependencies necessary for efficient planning and transferability.

Exploration improvements, such as intrinsic rewards based on information gain (Zhao et al., 2023), prediction errors (Burda et al., 2018), or visitation counts (Zheng et al., 2021; Wang et al., 2023b), enhance sample efficiency but lack structured reasoning for generalization. In contrast, Tsividis et al. (Tsividis et al., 2021) propose Theory-Based RL (TBRL), exemplified by EMPA (Exploring, Modeling, and Planning Agent), which integrates Bayesian causal modeling, structured exploration, and heuristic planning. EMPA achieves human-like efficiency by leveraging intuitive theories to generalize across diverse tasks with minimal training.

Cao et al. (Cao et al., 2022) introduce GALOIS (Generalizable Logic Synthesis) to bridge imperative and declarative programming paradigms for DRL. Using a hybrid sketch-based domain-specific language and differentiable inductive logic programming, GALOIS synthesizes interpretable, hierarchical programs with strict cause-effect logic. These white-box programs excel in generalizability and knowledge reuse across tasks but rely on predefined program sketches, limiting flexibility in loosely structured environments.

Symbolic approaches like STRIPS and Behavior Trees (BTs) (Guo et al., 2023; Colledanchise & Ögren, 2018) handle human-defined causal knowledge but lack adaptive learning capabilities. Similarly, POMDPs (Spaan, 2012) focus on probability updates rather than causal discovery, while causal networks (Pearl, 1995) and structure-learning methods (Zheng et al., 2018) face challenges with ambiguity and scalability.

While TBRL and GALOIS provide robust solutions for structured learning, their reliance on predefined logic or priors introduces complexity. Our approach proposes lightweight "empirical" causal relations learned from recurring cause-effect patterns, supporting efficient real-time learning in grid worlds without predefined program sketches or Bayesian modeling. This ensures adaptability while retaining interpretability and transferability.

### 2.1 Select Reinforcement Learning techniques in grid worlds

Deep Reinforcement Learning (DRL) often struggles with sample efficiency, requiring substantial interactions with environments. This paper examines foundational algorithms, scalable architectures, and exploration-focused methods that address these challenges, particularly in sparse-reward or complex environments like Minigrid.

Foundational methods include Deep Q-Network (DQN) Mnih et al. (2015), which combines deep neural networks with Q-learning to handle large state spaces but struggles with sparse rewards; Advantage Actor-Critic (A2C) Mnih et al. (2016), which reduces variance in updates through synchronized parallel actors but is limited by its on-policy nature; Trust Region Policy Optimization (TRPO) Schulman et al. (2015), which ensures stable policy updates with trust region constraints but is computationally intensive; and Proximal Policy Optimization (PPO) Schulman et al. (2017), which refines TRPO with clipped objectives for improved data utilization and computational efficiency.

Scalable architectures such as Importance Weighted Actor-Learner Architectures (IMPALA) Espeholt et al. (2018) address multi-task learning by leveraging distributed architectures with off-policy corrections, offering scalability but facing synchronization challenges, whereby exploration-focused methods aim to address sparse rewards and complex state spaces. Count-Based Exploration

(COUNT) Bellemare et al. (2016) uses pseudo-counts for better exploration but is computationally demanding in large state spaces. Random Network Distillation (RND) Burda et al. (2018) incentivizes novelty through prediction errors, however it depends on high-quality state representations. Curiosity-Driven Exploration (CURIOSITY) Pathak et al. (2017) rewards prediction errors of action outcomes, fostering intrinsic motivation, while Rewarding Impact-Driven Exploration (RIDE) Raileanu & Rocktäschel (2020) focuses on impactful actions but may struggle with ambiguous state changes. Adversarially Motivated Intrinsic Goals (AMIGo) Campero et al. (2021) generates adversarial goals to guide exploration, requiring robust goal generation mechanisms for effectiveness.

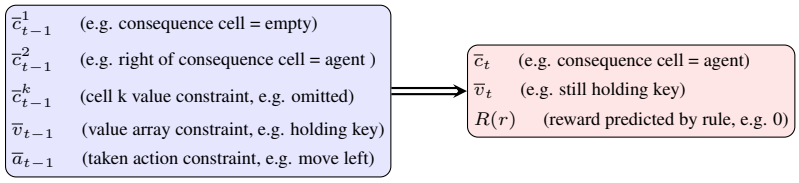
# 3 NON-AXIOMATIC CAUSAL EXPLORER

**NACE** is our proposed experiential learning technique with causality-informed intrinsic reward and strong inductive biases for grid world environments to boost sample efficiency further.

## 3.1 STATES AND RULE REPRESENTATION

State in NACE consists of a 2-dimensional value array and a one-dimensional value array. The 2-dimensional array reflects the spatial structure in the grid world (including remembered cells outside of the current view of the agent), while the one-dimensional array has no notion of spatial distance and is used only for internal values such as inventory items, e.g. keys, the agent can use or pick up.

Each rule is of the form $(preconditions, action) \Rightarrow consequence$ where the precondition can hold a conjunction of cell value constraints spatially relative to the cell value of the consequence, and the consequence predicts one particular cell's value as well as the values of the one-dimensional array at the next timestep.




Each rule tracks evidence using counters for $w_+$ and $w_-$ similar to (Wang, 2013), which measure the accuracy of the rule's predictions. Positive evidence ($w_+$) is accumulated whenever a perfectly matching rule predicts correctly, while negative evidence ($w_-$) increases with incorrect predictions. Tracking of evidence helps the agent refine its causal knowledge by prioritizing more reliable rules.

## 3.2 INDUCTIVE BIASES

It is well-known that favorable inductive biases can enhance sample efficiency. Below are inductive biases that are incorporated in NACE and relevant for many grid world environments:

1. **Temporal Locality**: NACE constructs rules based solely on the current and previous state, modeling relevant dependencies locally in time.
2. **Causal Representation**: NACE's knowledge representation is centered around the aforementioned causal rules which can be chained and are independent of the objective.
3. **Spatial Equivariance**: Ability to model causal dependencies between grid cells independently of the specific location of the cells considered in the dependency. This means learned rules in NACE can be applied at any location.
4. **State Tracking**: Ability to effectively track state outside of the field of view of the agent based on the recorded or estimated locations. NACE explicitly keeps track of a bird's-eye view map by recording observations into it, updating the values that are within its observability window.
5. **Attentional Bias**: Relevant dependencies tend to involve values that have either observably changed or a different value than predicted. Only rules that show a change from the previous to the current time step, or differ from the predicted value, are considered for rule formation, evidence updating, and prediction.

### 3.3 CURIOSITY MODEL

This section outlines the mechanism which helps NACE systematically acquire missing causal knowledge about the environment. The key principle is realized by making the agent plan to reach a state which it is most unfamiliar with. The familiarity is judged by whether existing rules match well to the situation, whereby matching is a matter of degree dependent on how many rule conditions match to the cells in the known state. This motivates the following formalism:

- *Match* value of a rule $r$ is evaluated relative to consequence cell $c$:

$$M(r, c) = \frac{\text{Number of matched preconditions}}{\text{Total number of preconditions}}$$

- *Cell match* value of a cell $c$ dependent on all $k$ existing rule match values:

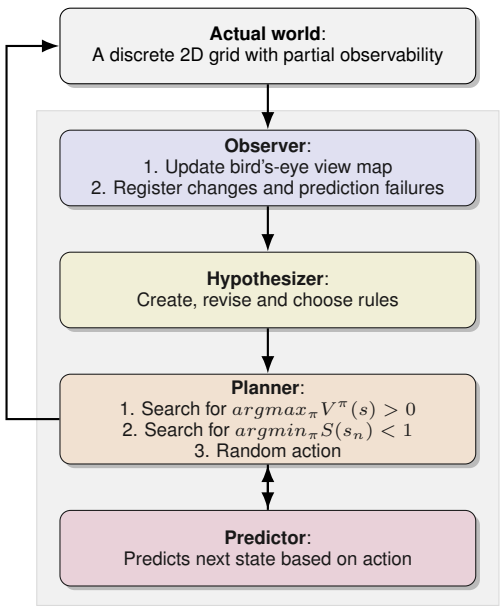
$$C(c) = \max(0, M(r_1, c), \ldots, M(r_k, c))$$

- *State match* value $S(s)$ of a state $s$ is the average $C(c)$ of its cells with $C(c) > 0$. This value, as we will see, is the secondary "explorative" objective in the planning process that guides the agent's decisions:

$$S(s) = \sum_{c \in X(s)} \frac{C(c)}{|X(s)|} \text{ where } X(s) = \{c \in s | C(c) > 0\},$$

### 3.4 NACE ARCHITECTURE

Figure 1 illustrates the high-level architecture of NACE, which consists of several interconnected components that work together to enable learning and decision-making, as follows:




**Actual world** represents the real simulated 2D grid environment (Minigrid) with cell-granular partial observability model. In each frame the field-of-view local to the agent is passed on to the observer.

**Observer** takes the field-of-view 2D array as input and and detects changes in values as well as identifies prediction failures from rules that predict incorrectly.

**Hypothesizer** creates and updates rules based on whether their predictions align with observations, whereby only changed-cells and prediction-mismatch cells as reported by Observer are considered.

**Planner** searches for optimal actions which lead to greater-than-zero expected return, and if none such is found, searches for actions which lead to a state of lowest state match value greater than zero. Finally, in case such also does not exist, a random action is chosen

**Predictor** forecasts the next state from the current state and the taken action, utilizing individual rules to predict a state transition of the entire state, whereby for each cell its predicted value comes from the rule with highest $M(r, c)$.

Figure 1: Flow diagram of the system

1. **Observer:** Its role is to update a bird's-eye view map via values from the partial observation 2D array, then to find changes in input, as well as prediction-observation mismatches (prediction failures). Formally this corresponds to determining the sets:
   - **Set of changes in observations**: $M_t^{\text{change}} = \{c_{t,x,y}^{\text{observation}} \mid c_{t,x,y}^{\text{observation}} \neq c_{t-1,x,y}^{\text{observation}}\}$ This set captures all grid cells $c_{x,y}$ where the observed value has changed between timesteps $t-1$ and $t$, highlighting areas that have been updated or modified.
   - **Set of observation mismatches**: $M_{\text{mismatched},t}^{\text{observation}} = \{c_{t,x,y}^{\text{observation}} \mid c_{t,x,y}^{\text{prediction}} \neq c_{t,x,y}^{\text{observation}}\}$ This set includes all grid cells where the observed value differs from the predicted value at time $t$, indicating potential prediction failures.

- **Set of prediction mismatches**: $M^{\text{prediction}}_{\text{mismatched},t} = \{c^{\text{prediction}}_{t,x,y} \mid c^{\text{prediction}}_{t,x,y} \neq c^{\text{observation}}_{t,x,y}\}$ This set identifies all grid cells where the predicted value does not match the observed value at time $t$, from the perspective of predictions.

These sets enable the Observer to track state changes and prediction failures, ensuring an accurate understanding of the environment and supporting the system's adaptive and predictive capabilities.

2. **Hypothesizer:** Associating positive and negative evidence based on prediction success, as well as creating new rules when positive evidence is found for the first time.

   Formally, for each rule $r = ((\overline{c}^1_{t-1} \wedge ... \wedge \overline{c}^k_{t-1} \wedge \overline{v}_{t-1} \wedge \overline{a}_{t-1}) \Rightarrow (\overline{c}_t \wedge \overline{v}_t \wedge R(r)))$, $\overline{c} := (c_r = c)$ indicates that the value in the rule precondition aligns with the actual cell value, value array in case of $\overline{v}$, and taken action in case of $\overline{a}$.

   The rule preconditions are met when all *equality constraints* $\overline{c}^1_{t-1}, ..., \overline{c}^k_{t-1}, \overline{v}_{t-1}, \overline{a}_{t-1}$ hold. Positive evidence is attributed when the equality constraints of the postcondition $\overline{c}_t, \overline{v}_t$ are met as well and the predicted reward aligns with the observed reward ($R_t = R(r)$):

$$w_+(r) = \begin{cases} w_+(r) + 1 & \text{if } \{c^1_{t-1}, ..., c^k_{t-1}, c_t\} \subseteq (M^{change}_t \cup M^{\text{observation}}_{\text{mismatched}_t}) \\ w_+(r) & \text{otherwise} \end{cases}$$

   Negative evidence is assigned when any of the postcondition equality constraints is not met:

$$w_-(r) = \begin{cases} w_-(r) + 1 & \text{if } c_t \in M^{\text{prediction}}_{\text{mismatched}_t} \\ w_-(r) & \text{otherwise} \end{cases}$$

   Finally, rules $r$ for which $w_-(r) > w_+(r)$ become inactive, and for two rules $r_1, r_2$ if their preconditions match (including the action) but the postconditions are different, only the rule with the higher truth expectation is selected, which is calculated according to:

$$w(r) = w_-(r) + w_+(r), \quad frequency(r) = \frac{w_+(r)}{w(r)}, \quad confidence(r) = \frac{w(r)}{w(r)+1}$$
$$f_{exp}(r) = (frequency(r) - \tfrac{1}{2}) * confidence(r) + \tfrac{1}{2}$$

   This not only allows the system to find the relevant preconditions under which a consequence happens when the action is utilized, but also gives the system tolerance to non-deterministic effects and enables accounting for uncertainty. A basic analysis of this can be found in the appendix.

3. **Planner**: NACE makes use of depth- and width-bounded Breadth-First-Search algorithm with a combined search objective consisting of two components: it searches for states resulting from the different action sequences for futures that lead to max. expected return or, if not existing, lowest state match value. Hence, it applies a key RL principle to maximize the expected long-term return (Sutton et al., 1999), with the policy determined by the considered action sequence: $\pi(t) = a_t \quad \text{for} \quad t = 1, 2, \ldots, n$ whereby $n$ is smaller-or-equal (dependent on where the optimum is found) to the maximum planning horizon:

$$\pi(t) = \begin{cases} \arg\max_\pi V^\pi(s_0) & \text{if } V^\pi(s_0) = \mathbb{E}\left[\sum_{t=0}^n \gamma^t R(s_t) \mid s_0 = s, \pi\right] > 0 \\ \arg\min_\pi S(s_n) < 1 & \text{otherwise} \end{cases}$$

   According to this definition, if no return greater than zero can be obtained for any considered action sequence, the system instead plans for a future state of lowest state match value, whereby $(\forall k : (0 \leq k < n) \rightarrow S(s_k) = 1) \wedge S(s_n) < 1$, meaning the action sequence is constrained to be planned in such a way that state match value is 1 except for the last action where it is minimized for the resulting state.

   Such constraint maximizes the agent's chance to reach the state of minimum state match while ensuring the low match value is not a consequence of predicting further from states where the knowledge was already not fully applicable.

   Due to the amount of possible options the planning algortihm dominates the asymptotics of NACE. It has the computational complexity of $O(|V| + |E|)$ where $V$ is the set of nodes, and $E$ the set of edges of the search graph. Constant-bounded search depth and width can be achieved by pruning of branches by expected return and state match value, however bounded search depth can negatively affect performance as analyzed in the appendix.

4. **Predictor**: When the planner queries for the predicted state from a given state and an action, the role of predictor is to construct the predicted state by applying all knowledge to the given state in the following way: initializing with the cell values from the given state, where for each cell we utilize only the rule $r$ with $M(r, c) = 1$ and maximum $f_{exp}(r)$, meaning the rule preconditions match perfectly to the given state, the action that has been considered, and $r$ has the highest truth expectation among the rule candidates.

   In this case the postcondition cell value of the rule is applied to the corresponding cell at position $(x, y)$ in the predicted state, while else the cell keeps the value from the previous state. Hence, for utilized rules $r^* = ((\overline{c}_t^1 \wedge ... \wedge \overline{c}_t^k \wedge \overline{v}_t \wedge \overline{a}_t) \Rightarrow (\overline{c}_{t+1} \wedge \overline{v}_{t+1} \wedge R(r^*)))$, where $\overline{c}_{t+1}$ and $\overline{v}_{t+1}$ constrains the cell value and value array of the consequence:

$$c_{t+1,x,y} = \begin{cases} c_{t+1} & \text{if } r^* = \underset{r|M(r,c_{t,x,y})=1}{\text{argmax}} f_{\text{exp}}(r) \\ c_{t,x,y} & \text{otherwise} \end{cases}$$

   Now, while $s_{t+1}$ is a composition of the cells at all locations at time $t + 1$, the reward associated with $s_{t+1}$ is the average of the reward of each of the $N$ utilized rules:

$$R(s_{t+1}) = \frac{1}{N} \sum_{i=1}^{N} R(r_i^*)$$

# 4 Experiments in Minigrid

To evaluate the effectiveness of NACE compared to other Deep Reinforcement Learning (DRL) techniques, we conducted a series of experiments in Minigrid Chevalier-Boisvert et al. (2024), a 2D grid world environment featuring diverse and procedurally generated scenarios. We focus on Minigrid levels that feature partial observability (using the default observation format which provides values per grid cell rather than per pixel), challenging the agent to operate with limited information about its surroundings. The selected environments (see Table 1) are categorized based on the specific challenges they present:

1. **Static:** fixed start & goal locations.
2. **Dynamic:** randomized start, goal, and obstacle positions.
3. **Dynamic with sequential dependencies:** tasks requiring specific action sequences (e.g., a door that needs a key or switch to be opened).

| Environment | Type |
|---|---|
| MiniGrid-Empty-16x16-v0 | 1 |
| MiniGrid-DistShift2-v0 | 1 |
| MiniGrid-LavaGapS7-v0 | 2 |
| MiniGrid-SimpleCrossingS11N5-v0 | 2 |
| MiniGrid-Unlock-v0 | 3 |
| MiniGrid-DoorKey-8x8-v0 | 3 |

Table 1: Environments with corresponding types

In each environment, we recorded the average reward, episode length, and standard deviation every 100 time steps, whereby each time step incorporates the observed state, action taken and obtained reward. The following sections present and discuss some representative results for each category, using the select reinforcement learning techniques mentioned in section 2.1. Additionally, Behavior Trees (BTs) and hard-coded policies are employed as performance upper bounds in non-stationary and static environments.

## 4.1 Stationary Environments

In this category, because of the start and goal locations are fixed, the primary challenge for the agent is to consistently learn and optimize navigation strategies over repeated episodes.

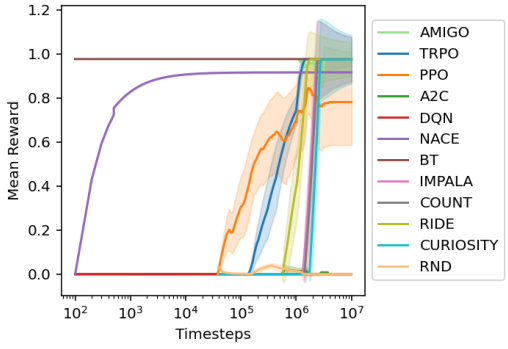



Figure 2: Learning curves in *MiniGrid-Empty-16x16-v0*

| Techn. | Avg. reward | S. dev. |
|---|---|---|
| **TRPO** | **0.976** | **0.001** |
| **PPO** | 0.781 | 0.390 |
| **A2C** | 0.000 | 0.000 |
| **DQN** | 0.000 | 0.000 |
| **IMPALA** | **0.976** | **0.197** |
| **COUNT** | 0.974 | 0.203 |
| **RIDE** | 0.975 | 0.151 |
| **CURIOSITY** | 0.974 | 0.217 |
| **RND** | 0.000 | 0.008 |
| **AMIGO** | **0.976** | **0.059** |
| **NACE** | 0.916 | 0.004 |

Table 2: Final values for *MiniGrid-Empty-16x16-v0*

**MiniGrid-Empty-16x16-v0:** This environment features a large, static grid where the agent must navigate from a fixed start location to a fixed goal. Due to the limited observation window and the sparse reward—only granted upon reaching the goal—the task can present some difficulties. In this case, NACE, following its innate strategy, first learns to move effectively within the grid by exploring its immediate surroundings. Then it systematically expands its exploration, hitting observed walls out of curiosity, and finally exploring initially invisible parts of the map until the goal object is found and moved into. Due to its intrinsic inductive bias, it explores the area systematically and associates reward with the goal location within about $10^3$ time steps. In contrast, other techniques like TRPO, IMPALA, RIDE, and AMIGO, although capable of eventually learning the task, require over $10^5$ time steps to perform comparatively as well (as seen in Figure 2, and Table 2).

**MiniGrid-DistShift2-v0:** In this case, the fixed start and goal locations are accompanied by stationary lava obstacles, which the agent must navigate around to reach the goal. DQN performs quite well, achieving a near-optimal policy with an average reward of $0.96$, closely mirroring the performance of the BT. Although reaching a slightly lower average reward of $0.87$, NACE was three orders of magnitudes more sample-efficient. The next-best policies were found by AMIGO and PPO with an average reward of $0.78$ and $0.76$, while the other techniques were below $0.5$, all of them being much less sample-efficient than NACE (as in Figure 3 and Table 3).

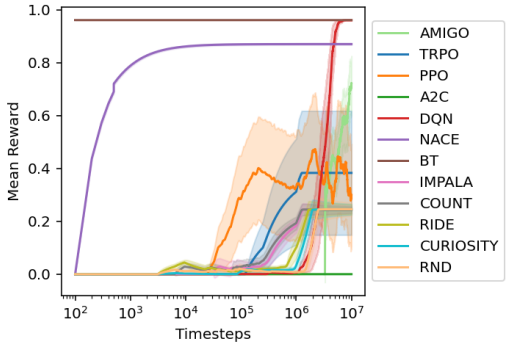



Figure 3: Learning curves in *MiniGrid-DistShift2-v0*

| Techn. | Avg. reward | S. dev. |
|---|---|---|
| **TRPO** | 0.383 | 0.469 |
| **PPO** | 0.763 | 0.381 |
| **A2C** | 0.000 | 0.000 |
| **DQN** | **0.961** | **0.000** |
| **IMPALA** | 0.245 | 0.027 |
| **COUNT** | 0.243 | 0.025 |
| **RIDE** | 0.245 | 0.036 |
| **CURIOSITY** | 0.245 | 0.041 |
| **RND** | 0.245 | 0.049 |
| **AMIGO** | 0.778 | 0.203 |
| **NACE** | 0.870 | 0.006 |

Table 3: Final values for *MiniGrid-DistShift2-v0*

### 4.2 Dynamic Environments

Given that the start and goal locations, along with obstacle positions, are randomized in each episode, these environments require the agent to continuously adapt to new and unpredictable conditions.

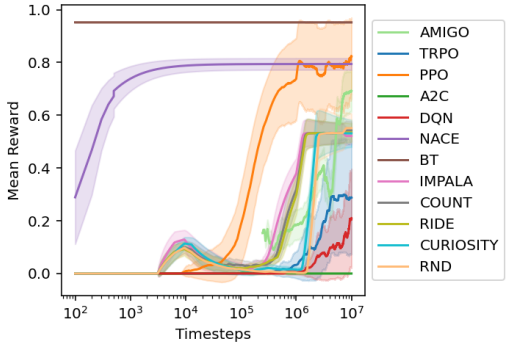



Figure 4: Learning curves in *MiniGrid-LavaGapS7-v0*

| Techn. | Avg. reward | S. dev. |
|---|---|---|
| **TRPO** | 0.187 | 0.375 |
| **PPO** | **0.838** | **0.309** |
| **A2C** | 0.000 | 0.000 |
| **DQN** | 0.114 | 0.309 |
| **IMPALA** | 0.521 | 0.064 |
| **COUNT** | 0.543 | 0.067 |
| **RIDE** | 0.535 | 0.070 |
| **CURIOSITY** | 0.531 | 0.103 |
| **RND** | 0.551 | 0.111 |
| **AMIGO** | 0.690 | 0.151 |
| **NACE** | 0.794 | 0.044 |

Table 4: Final values for *MiniGrid-LavaGapS7-v0*

**MiniGrid-LavaGapS7-v0:** In this environment, the agent must navigate around randomly placed lava obstacles to reach a fixed goal, requiring adaptability due to the varying paths between episodes. The 5x5 free space, which is mostly covered by the agent's observation window, is being complicated by dynamically spawning lava, unlike the stationary obstacles in *MiniGrid-DistShift2-v0*. From the mean episode rewards (as seen in Figure 4, and Table 4), it is clear that PPO and NACE find a similarly effective strategy, whereby NACE takes about $10^3$ time steps while PPO takes $3 \times 10^5$ time steps to reach a mean reward value of around $0.8$, while the optimal policies, as BT shows, vary between $0.9$ and $1.0$. Additionally, PPO exhibits more instability in learning and greater sensitivity to initialization, as evidenced by a higher standard deviation. Following these, AMIGO reached an average reward of only $0.69$, while the remaining techniques performed poorly, despite the fact that this level is practically fully observable.

**MiniGrid-SimpleCrossingS11N5-v0:** Here the agent faces a large grid with multiple intersections and potential dead ends. The randomized layout in each episode also forces the agent to develop a robust exploration strategy. As Figure 5, and Table 5 show, IMPALA, COUNT, RIDE, CURIOSITY, RND achieved near-optimal policies, since their intrinsic reward mechanisms seem to be particularly helpful in this large environment where the observable window covers only a small part. NACE and AMIGO found reasonable policies with average rewards of $0.88$ and $0.78$ respectively, while the remaining techniques scored below $0.5$. Again, NACE's strength lies in its data efficiency, driven by its inductive biases, even though it does not converge to the optimal policy.

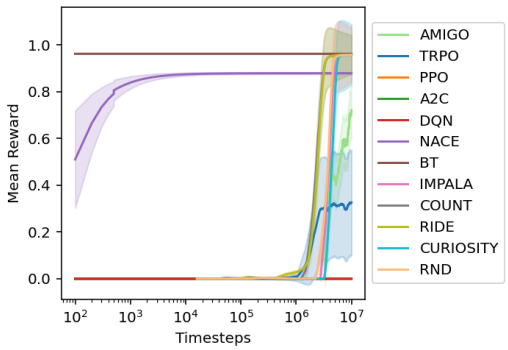



Figure 5: Learning curves in *MiniGrid-SimpleCrossingS11N5-v0*

| Techn. | Avg. reward | S. dev. |
|---|---|---|
| **TRPO** | 0.381 | 0.467 |
| **PPO** | 0.000 | 0.000 |
| **A2C** | 0.000 | 0.000 |
| **DQN** | 0.000 | 0.000 |
| **IMPALA** | 0.958 | 0.238 |
| **COUNT** | **0.960** | **0.168** |
| **RIDE** | 0.959 | 0.170 |
| **CURIOSITY** | 0.958 | 0.261 |
| **RND** | 0.958 | 0.222 |
| **AMIGO** | 0.778 | 0.203 |
| **NACE** | 0.880 | 0.009 |

Table 5: Final values for *MiniGrid-SimpleCrossingS11N5-v0*

### 4.3 Dynamic Environments with Sequential Dependencies

In these environments, the need to perform actions in a specific sequence adds complexity and tests the agent's ability to plan and execute multi-step strategies.

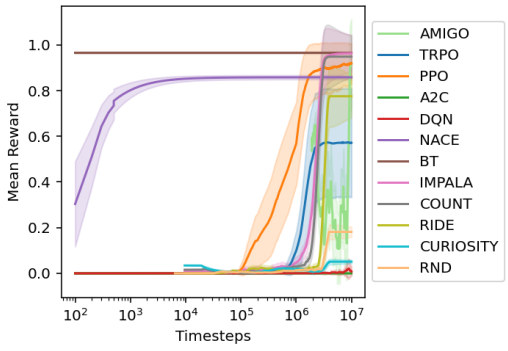



Figure 6: Learning curves in *MiniGrid-Unlock-v0*

| Techn. | Avg. reward | S. dev. |
|---|---|---|
| **TRPO** | 0.577 | 0.471 |
| **PPO** | 0.890 | 0.263 |
| **A2C** | 0.000 | 0.000 |
| **DQN** | 0.000 | 0.000 |
| **IMPALA** | **0.964** | **0.162** |
| **COUNT** | 0.949 | 0.185 |
| **RIDE** | 0.775 | 0.188 |
| **CURIOSITY** | 0.051 | 0.016 |
| **RND** | 0.181 | 0.046 |
| **AMIGO** | 0.932 | 0.388 |
| **NACE** | 0.858 | 0.018 |

Table 6: Final values for *MiniGrid-Unlock-v0*

**MiniGrid-Unlock-v0:** In this scenario, the agent must first locate and pick up a key before unlocking a door to reach the goal and obtain the reward. This sequential dependency adds a layer of complexity that challenges the agent's ability to plan ahead. Even though it is a single sequential dependency, the DRL technique that learned the fastest initially, PPO, demands about a million time steps to converge to a similarly effective policy as NACE, which achieves this within just $10^3$ steps again (as seen in Figure 6, and Table 6). Additionally, while PPO shows more instability in learning, it is far less chaotic than AMIGO. IMPALA reached the optimal policy, and did so after about 2 million steps, performing similarly well as COUNT and AMIGO in the end. It is also visible, in our runs, that TRPO did not exceed a mean episode reward of $0.6$, while A2C and DQN completely failed to learn any effective policy.

**MiniGrid-DoorKey-8x8-v0:** This environment introduces an additional layer of sequential dependency by requiring the agent to navigate through an unlocked door to reach a goal in a separate room. While passing through the door adds complexity, the primary challenge lies in the sparse reward structure, as no reward is given for merely using the door, since only reaching the final goal is rewarded. As presented in Figure 7 and Table 7, COUNT nearly achieved the optimal policy with an average reward of $0.96$. AMIGO reached $0.87$, but within $10^7$ time steps, which is below the average reward which NACE reached within only $10^4$ steps. Overall the results suggest poor combinatorial scaling of the involved DRL techniques, while NACE, on average, required a similar amount of time steps as for *MiniGrid-Unlock-v0* to learn an effective policy.

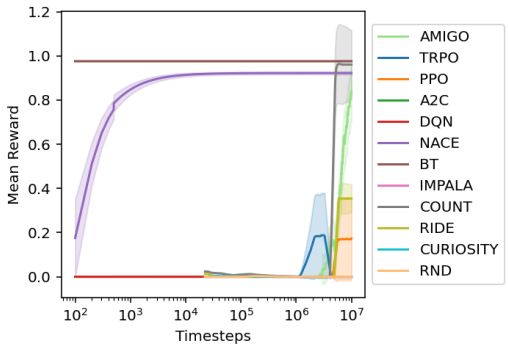



Figure 7: Learning curves in *MiniGrid-DoorKey-8x8-v0*

| Techn. | Avg. reward | S. dev. |
|---|---|---|
| **TRPO** | 0.000 | 0.000 |
| **PPO** | 0.156 | 0.357 |
| **A2C** | 0.000 | 0.000 |
| **DQN** | 0.000 | 0.000 |
| **IMPALA** | 0.000 | 0.000 |
| **COUNT** | **0.960** | **0.308** |
| **RIDE** | 0.354 | 0.126 |
| **CURIOSITY** | 0.000 | 0.000 |
| **RND** | 0.000 | 0.001 |
| **AMIGO** | 0.868 | 0.241 |
| **NACE** | 0.922 | 0.012 |

Table 7: Final values for *MiniGrid-DoorKey-8x8-v0*

### 4.4 Comparison with a model-based technique

The previous comparison focused on model-free methods with intrinsic rewards without modeling state transition dynamics. Model-based RL techniques can capture explicit state transitions

which include the entire state, or if the state is compositional, can incorporate transitions between parts of states such as the rules in NACE. Due to inherent variability, full state transitions are usually not practical in case of high-dimensional spaces, and the latter is restricted to domains where state composition can easily be achieved via various feature extraction techniques, or when it is straightforward to partition states as in grid world environments. If there is higher variability in higher-dimensional spaces, a latent dynamics model can be learned Hafner et al. (2020) instead of representing explicit transitions. This approach, as employed by DreamerV2, offers broad applicability beyond grid worlds and is well-suited for domains with high-dimensional state spaces. The only disadvantage of the latter is that the latent space representation has to be learned in addition to its dynamics, which increases the data demand. This is in line with the quantitative results we collected for selected environments from the literature Hong et al. (2022), whereby we show the converged performance as well as when NACE reaches the performance DreamerV2 converges to from our runs:

| Technique | Environment | Average Reward | Time Steps |
|---|---|---|---|
| NACE | DoorKey-6x6-v0 | 0.89 | 3.00E+02 |
| NACE | DoorKey-6x6-v0 | 0.93 | 1.00E+05 |
| DreamerV2 | DoorKey-6x6-v0 | 0.89 | 1.00E+05 |
| NACE | Unlock-v0 | 0.60 | 2.70E+02 |
| NACE | Unlock-v0 | 0.86 | 2.50E+06 |
| DreamerV2 | Unlock-v0 | 0.60 | 2.50E+06 |

Table 8: Comparison of NACE and DreamerV2

These results indicate that NACE's higher sample efficiency can be explained by its processing exploiting the cell-based grid world state observations. While this represents a strong inductive bias by itself, it can potentially be valuable as the mapping from high-dimensional input to a similar discrete world representation, can dependent on the problem domain be feasible and potentially be implemented with the proper choice of feature extraction techniques. This is commonly addressed in robotics to operate mobile robots Zhang et al. (2020a), typically by integrating Simulatenous Localization and Mapping (SLAM) and object detection models to construct semantic maps.

## 5 Conclusion

In this paper, we introduced NACE, an experiential learning agent designed to enhance data efficiency in grid world environments by leveraging causally-informed intrinsic rewards and inductive biases. We compared NACE with state-of-the-art DRL techniques, demonstrating that while these techniques are able to eventually achieve near-optimal policies, they often require significantly more data, especially as task complexity increases due to factors such as sequential dependencies. NACE, by contrast, extends the RL framework to empirically support causal relations, enabling effective learning and decision-making even in low data settings without relying on pre-defined causal models. Our causality-informed curiosity model, combined with the outlined inductive biases, facilitates systematic exploration and learning requiring significantly fewer time steps. We hope that future work in the field will strike new compromises regarding the inclusion of inductive biases, leading to highly sample-efficient DRL that retains the ability to converge to optimal policies. Moving forward, we plan to generalize NACE to handle three-dimensional and continuous spaces, as well as explore neural implementations of NACE for causality-informed Deep RL, further advancing the capabilities of learning agents in complex environments.

## Reproducibility Statement

- We utilized open-source implementations of the selected DRL algorithms from public repositories (not including our technique):
  - AMIGo was from here: https://github.com/facebookresearch/adversarially-motivated-intrinsic-goals
  - BT is here: https://github.com/andreneco/minigrid_bt
  - DQN, A2C, TRPO, and PPO were established on Stable Baselines3 (SB3)'s baselines repository (Raffin et al., 2021): https://stable-baselines3.readthedocs.io/

  - All the other were from here: https://github.com/sparisi/cbet
- We used the the Minigrid package for the environments in our comparison, which is available here: https://github.com/Farama-Foundation/Minigrid
- For NACE we provide a stand-alone zip archive for reviewers to reproduce our results, which is runnable on a regular computer with Python interpreter. It includes a README.txt in the NACE folder, as well as scripts to generate the tables and the plots present in the paper.

## REFERENCES


Nishanth Anand and Doina Precup. Prediction and control in continual reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Daniel Bryce. Wumpus world in introductory artificial intelligence. *Journal of Computing Sciences in Colleges*, 27(2):58–65, 2011.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

A Campero, R Raileanu, H Küttler, JB Tenenbaum, T Rocktäschel, and E Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. In *ICLR 2021-9th International Conference on Learning Representations*. International Conference on Learning Representations, 2021.

Yushi Cao, Zhiming Li, Tianpei Yang, Hao Zhang, Yan Zheng, Yi Li, Jianye Hao, and Yang Liu. Galois: boosting deep reinforcement learning via generalizable logic synthesis. *Advances in Neural Information Processing Systems*, 35:19930–19943, 2022.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Huihui Guo, Fan Wu, Yunchuan Qin, Ruihui Li, Keqin Li, and Kenli Li. Recent trends in task and motion planning for robotics: A survey. *ACM Computing Surveys*, 55(13s):1–36, 2023.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Zhang-Wei Hong, Tao Chen, Yen-Chen Lin, Joni Pajarinen, and Pulkit Agrawal. Topological experience replay. *arXiv preprint arXiv:2203.15845*, 2022.

Woojun Kim, Yongjae Shin, Jongeui Park, and Youngchul Sung. Sample-efficient and safe deep reinforcement learning via reset deep ensemble agents. *Advances in Neural Information Processing Systems*, 36, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Judea Pearl. From bayesian networks to causal networks. In *Mathematical models for handling partial knowledge in artificial intelligence*, pp. 157–182. Springer, 1995.

Martha E Pollack and Marc Ringuette. Introducing the tileworld: Experimentally evaluating agent architectures. In *AAAI*, volume 90, pp. 183–189, 1990.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*, 2020.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Matthijs TJ Spaan. Partially observable markov decision processes. *Reinforcement learning: State-of-the-art*, pp. 387–414, 2012.

Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

Pedro A Tsividis, Joao Loula, Jake Burga, Nathan Foss, Andres Campero, Thomas Pouncy, Samuel J Gershman, and Joshua B Tenenbaum. Human-level reinforcement learning through theory-based modeling, exploration, and planning. *arXiv preprint arXiv:2107.12544*, 2021.

Andrew Wang, Andrew C Li, Toryn Q Klassen, Rodrigo Toro Icarte, and Sheila A McIlraith. Learning belief representations for partially observable deep rl. In *International Conference on Machine Learning*, pp. 35970–35988. PMLR, 2023a.

Kaixin Wang, Kuangqi Zhou, Bingyi Kang, Jiashi Feng, and YAN Shuicheng. Revisiting intrinsic reward for exploration in procedurally generated environments. In *The Eleventh International Conference on Learning Representations*, 2023b.

Pei Wang. *Non-axiomatic logic: A model of intelligent reasoning*. World Scientific, 2013.

Jiadong Zhang, Wei Wang, Xianyu Qi, and Ziwei Liao. Social and robust navigation for indoor robots based on object semantic grid and topological map. *Applied Sciences*, 10(24):8991, 2020a.

Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*, 2020b.

Qian Zhao, Jinhui Han, and Mao Xu. Boosting policy learning in reinforcement learning via adaptive intrinsic reward regulation. *IEEE Access*, 2023.

Lulu Zheng, Jiarui Chen, Jianhao Wang, Jiamin He, Yujing Hu, Yingfeng Chen, Changjie Fan, Yang Gao, and Chongjie Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. *Advances in Neural Information Processing Systems*, 34:3757–3769, 2021.

Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. Dags with no tears: Continuous optimization for structure learning. *Advances in neural information processing systems*, 31, 2018.

## APPENDIX A ADDITIONAL STUDIES AND DISCUSSIONS

The performance results indicate that NACE often exhibits sub-optimal outcomes. To analyze this, we present contributing factors from a conceptual perspective, examine the impact of hyperparameter choices, and assess robustness to non-determinism arising from random action consequences.

- **Representational limitations**: NACE's rule-based framework captures only spatially relative dependencies from one time step to the next. It does not exploit the inherent structural statistics of environment generation, which are leveraged by various deep reinforcement learning (DRL) techniques. While these structural dependencies are most apparent in static environments where locations remain constant, they are also present in dynamic environments. For example, the goal location consistently appears in the bottom-right corner not only in *MiniGrid-Empty* levels but also in *MiniGrid-SimpleCrossingS11N5-v0*. NACE's inability to utilize these broader environmental patterns limits its performance compared to methods that can.
- **Study of reduced Planning Horizon**: NACE's estimation of expected returns relies heavily on its planning horizon. Short planning horizons can significantly reduce performance, especially in tasks requiring long-term planning. To quantify this effect, we examine two cases: *MiniGrid-DoorKey-8x8-v0*, which demands longer-horizon planning, and *MiniGrid-DoorKey-6x6-v0*, which is less demanding in this regard. As shown in Table 9, running NACE with a planning horizon of only 8 steps in *MiniGrid-DoorKey-8x8-v0* results in a convergence to an average return of 0.48, whereas extending the horizon to 100 steps improves the average reward to 0.92. In contrast, in *MiniGrid-DoorKey-6x6-v0*, NACE maintains an average reward of 0.93 regardless of the planning horizon. A similar pattern is observed in *MiniGrid-Empty-16x16-v0*, where the average reward drops from 0.91 to 0.41 when the planning horizon is reduced from 100 to 8 steps. These results highlight NACE's dependence on adequate planning horizons for effective rule chaining, and the significant performance degradation that occurs when the planning horizon is too short.

| Environment | Planning Horizon, Average Reward |
|---|---|
| MiniGrid-DoorKey-6x6-v0 | 8 steps, 0.93 |
| MiniGrid-DoorKey-6x6-v0 | 100 steps, 0.93 |
| MiniGrid-DoorKey-8x8-v0 | 8 steps, 0.48 |
| MiniGrid-DoorKey-8x8-v0 | 100 steps, 0.92 |
| MiniGrid-Empty-16x16-v0 | 8 steps, 0.41 |
| MiniGrid-Empty-16x16-v0 | 100 steps, 0.91 |

Table 9: NACE Performance with different planning horizons

- **Robustness to Non-Determinism**: NACE's rule representation incorporates uncertainty handling through evidence counters, enabling it to cope with non-deterministic state transitions. To assess this capability, we modify the environment to invoke unintended actions with certain probabilities. In *MiniGrid-Empty-16x16-v0*, when 10% of actions result in unintended outcomes, NACE still achieves an average reward above 0.9, demonstrating basic tolerance to non-determinism. However, when the probability of unintended actions increases to 20%, NACE fails to complete the task within the maximum allowed time in all episodes. Higher tolerance to non-determinism can be achieved by increasing the default truth expectation threshold for rule usage above the default value of 0.5. However, this adjustment reduces sample efficiency, as it requires the agent to confirm each rule multiple times before utilizing it.

- **Ablation discussion: omission of inductive biases**
  - The causal rule representation is a core part of NACE's operating and hence cannot be omitted. However planning horizon can be reduced to show the consequences of reduced chaining, as analyzed in the study of reduced planning horizon in this appendix.
  - The temporal locality and attentional bias cannot be omitted for realistic environment sizes, due to the combinatoric explosion of rules which would result from considering all cell combinations and across multiple time steps. The powerset of $w*h$ cells would be $2^{w*h}$ rule items if all cell combinations would be considered even in just a single step, and would be $2^{w*h*d}$ for a time window of duration $d$.
  - Omitting NACE state tracking would be problematic as it has no memory of prior states. In most instances it would be drawn to turn around indefinitely due to the curiosity model leading to low state match value for the turning around of the agent (as this area is always unknown without state tracking).
  - Last, the sample efficiency of NACE would suffer if the agent could not apply its rules at the different locations. Without spatial equivariance, each rule has to be learned for each encountered location, which dependent on the dimensions of the environment, e.g. with size $(w, h)$, would increase the necessary sample amount by a factor of $w * h$, which would eliminate the data efficiency gains of the approach.

## APPENDIX B NOTATION AND SYMBOLS

| Symbol | Description |
|---|---|
| $s$ | State, represented as a combination of a 2D grid ($s_{grid}$) and a 1D array ($s_{array}$) |
| $a$ | Action taken by the agent |
| $r$ | Causal rule in the form (preconditions, action) $\Rightarrow$ consequence |
| $c_{x,y}$ | Cell value at position $(x, y)$ in the 2D grid |
| $\overline{c}$ | Equality constraint on a cell value (e.g., $c_r = c$) |
| $M(r, c)$ | Match value of a rule $r$ for cell $c$, based on the fraction of preconditions satisfied |
| $C(c)$ | Cell match value for cell $c$, derived from the maximum match value across all rules |
| $S(s)$ | State match value for state $s$, calculated as the average $C(c)$ for cells with $C(c) > 0$ |
| $w_+(r)$ | Positive evidence counter for rule $r$, incremented when predictions align with observations |
| $w_-(r)$ | Negative evidence counter for rule $r$, incremented when predictions differ from observations |
| $w(r)$ | Total evidence count for rule $r$, defined as $w(r) = w_+(r) + w_-(r)$ |
| $f(r)$ | Fraction of positive evidence for rule $r$, defined as $f(r) = \frac{w_+(r)}{w(r)}$ |
| $c(r)$ | Confidence factor for rule $r$, defined as $c(r) = \frac{w(r)}{w(r)+1}$ |
| $f_{\text{exp}}(r)$ | Expected truth value for rule $r$, calculated as $f_{\text{exp}}(r) = (f(r) - \frac{1}{2}) \cdot c(r) + \frac{1}{2}$ |
| $M_t^{\text{change}}$ | Set of cells with changes in observed values between timesteps $t-1$ and $t$ |
| $M_{\text{mismatched},t}^{\text{observation}}$ | Set of cells where observed values differ from predicted values at timestep $t$ |
| $M_{\text{mismatched},t}^{\text{prediction}}$ | Set of cells where predictions differ from observations at timestep $t$ |
| $R(r)$ | Reward associated with rule $r$ |
| $R(s)$ | Reward associated with a state $s$, defined as the average reward of rules applied to generate $s$ |
| $V(s)$ | Value of state $s$, used in planning for maximizing long-term returns |
| $\pi(t)$ | Planned action sequence or policy at timestep $t$ |
| $\gamma$ | Discount factor for future rewards |

## APPENDIX C PSEUDOCODE

The system can be described by the pseudocode:

Algorithm 1: Pseudocode of NACE

- **Actual World:** perceived_array = perceive_partial(world)
- **Observer:**
  $s_t$ = update_bird_view($s_{t-1}$, perceived_array)
  $calculate(M_{change}, M^{observation}_{mismatched}, M^{prediction}_{mismatched})$
- **Hypothesizer:**
  - Create new rules for which $w_+(r) = 1$.
  - Update rule evidences according to $w_+(r)$ and $w_-(r)$.
  - Choose rules $r_1$ with $w_+(r_1) > w_-(r_1)$ for which there does not exist a rule $r_2$ with same precondition and action, but different postcondition with $f_{exp}(r_2) > f_{exp}(r_1)$.
- **Planner utilizing Predictor:**
  $a_1, ..., a_n = BFS_with_Predictor(V(s) > 0)$
  $a^*_1, ..., a^*_n = BFS_with_Predictor(min(S(s)) < 1)$
  //whereby BFS_with_Predictor is bounded breadth first search with Predictor as state transition function
  If found($a_1, ..., a_n$):, return $a_1, ..., a_n$
  If found($a^*_1, ..., a^*_n$):, return $a^*_1, ..., a^*_n$
  Else Perform a random action

# APPENDIX D HYPERPARAMETER DETAILS

## D.1 FOUNDATIONAL ALGORITHMS

### D.1.1 CORE MODELS AND THEIR MECHANISMS

- **Deep Q-Network (DQN):** DQN integrates deep neural networks with classical Q-learning, making it effective for handling large state spaces. To stabilize training, DQN uses experience replay and a separate target network. The Q-value update in DQN follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$$

  where:
  - $s, a$: current state and action,
  - $s', a'$: next state and action,
  - $R(s)$: reward received,
  - $\gamma$: discount factor for future rewards,
  - $\alpha$: learning rate.
- **Advantage Actor-Critic (A2C):** A2C builds on the actor-critic framework, synchronizing multiple parallel learners to reduce variance in policy updates. It calculates an **advantage function** to evaluate actions relative to the current policy's value estimate, stabilizing training but requiring frequent environmental interactions due to its on-policy nature.

  **Advantage Function:**

$$A(s,a) = Q(s,a) - V(s)$$

  **Policy Update:** The policy is updated using the gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) A(s,a)$$

  where:
  - $Q(s,a)$: action-value function,
  - $V(s)$: state-value function,
  - $\pi_\theta(a|s)$: policy parameterized by $\theta$,
  - $\alpha$: learning rate.

- **Trust Region Policy Optimization (TRPO):** TRPO addresses stability in policy updates by enforcing a trust region constraint, ensuring small policy changes during optimization. This constraint is implemented via a KL-divergence bound, preventing drastic shifts in behavior but requiring computationally expensive second-order optimization.

  **Objective Function:**

$$\max_{\theta} \mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}} \left[ \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A(s,a) \right]$$

  **Constraint:**

$$\mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}} \left[ D_{\text{KL}}(\pi_{\theta_{\text{old}}} || \pi_\theta) \right] \leq \delta$$

  where:

  - $\pi_\theta(a|s)$: new policy,
  - $\pi_{\theta_{\text{old}}}(a|s)$: previous policy,
  - $A(s,a)$: advantage function,
  - $D_{\text{KL}}$: KL-divergence,
  - $\delta$: trust region size.

- **Proximal Policy Optimization (PPO):** PPO refines TRPO by introducing a clipped surrogate objective, which simplifies computation and allows for multiple updates per batch. This approach improves data utilization while maintaining policy stability.

  **Clipped Surrogate Objective:**

$$\max_{\theta} \mathbb{E}_{s,a} \left[ \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A(s,a), \text{clip} \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon \right) A(s,a) \right) \right]$$

  where:

  - $\pi_\theta(a|s)$: new policy,
  - $\pi_{\theta_{\text{old}}}(a|s)$: old policy,
  - $A(s,a)$: advantage function,
  - $\epsilon$: clipping threshold.

#### D.1.2 Hyperparameter Configuration for Foundational Algorithms

We utilize the Stable Baselines3 framework (Raffin et al., 2021) to train and evaluate foundational algorithms, leveraging its pre-implemented models and customizable configurations. All algorithms use the same convolutional neural network architecture to process observations, ensuring consistency across experiments. The hyperparameters for each algorithm were selected based on achieving the best average performance across all tasks, rather than optimizing for a single task, to ensure generalizability. The details of the network architecture and training setup for each algorithm are outlined below.

- **Network Architecture:** Observations ($7 \times 7 \times 3$) from the MiniGrid environment are processed through four convolutional layers. Each layer is configured as follows:
  - Kernel size: $2 \times 2$
  - Activation: ReLU
  - Increasing number of filters: 16, 32, 64, and 128

  The output of the final convolutional layer is flattened and passed to a fully connected layer with:
  - Output dimension: 128
  - Activation: ReLU
- **Training Setup for DQN:**
  - Learning rate: $0.0001$
  - Buffer size: $1,000,000$
  - Learning starts: $100$
  - Batch size: $32$

  - Soft update coefficient: $1$
  - Discount factor: $0.99$
  - Train frequency: $4$
  - Gradient steps: $1$
  - Target update interval: $10,000$
  - Exploration fraction: $0.1$
  - Initial exploration epsilon: $1.0$
  - Final exploration epsilon: $0.05$
  - Max gradient norm: $10.0$
- **Training Setup for A2C:**
  - Learning rate: $0.0007$
  - Number of steps: $5$
  - Discount factor: $0.99$
  - Entropy coefficient: $0.0$
  - Value function coefficient: $0.5$
  - Max gradient norm: $0.5$
- **Training Setup for TRPO:**
  - Learning rate: $0.001$
  - Number of steps: $2048$
  - Batch size: $128$
  - Discount factor: $0.99$
  - Conjugate gradient max steps: $15$
  - Conjugate gradient damping: $0.1$
  - Line search shrinking factor: $0.8$
  - Line search max iterations: $10$
  - Number of critic updates: $10$
  - Target KL divergence: $0.01$
- **Training Setup for PPO:**
  - Learning rate: $0.0003$
  - Number of steps: $2048$
  - Batch size: $64$
  - Number of epochs: $10$
  - Discount factor: $0.99$
  - Clip range: $0.2$
  - Entropy coefficient: $0.0$
  - Value function coefficient: $0.5$
  - Max gradient norm: $0.5$

### D.2 Exploration and Scalability Extensions

All experiments for the other methods are based on the Torchbeast implementation of IMPALA (Espeholt et al., 2018), modified for intrinsic reward algorithms by Raileanu & Rocktäschel (2020) and Campero et al. (2021). Shared hyperparameters are listed first, followed by specific details for each algorithm.

#### D.2.1 Shared Hyperparameters

- **Network Architecture:** Observations ($7 \times 7 \times 3$ for MiniGrid) are processed through three convolutional layers:
  - 32 filters each
  - Kernel size: $3 \times 3$

  - Stride: 2
  - Padding: 1

  Each layer uses exponential linear unit (ELU) activation. Outputs are passed to a fully connected layer with 256 units, followed by:

  - A final fully connected layer for policy logits and value function (IMPALA), or
  - Auxiliary modules for intrinsic rewards (e.g., forward/inverse dynamics for RIDE and CURIOSITY).

- **Training Setup:**
  - Number of actors: 40
  - Number of buffers: 80
  - Unroll length: 100
  - Number of learner threads: 4
  - Batch size: 32
  - Discount factor: 0.99
  - Learning rate: 0.0001
  - Policy entropy loss: 0.0005
  - Gradient clipping: Norm of 40
  - Save interval: Every 20 minutes
- **Special Parameters (Only When Applicable):**
  - Count reset probability: 0.001 (COUNT, RIDE)
  - Hash bits: 128 (COUNT)

#### D.2.2 Intrinsic Rewards and Coefficients

Intrinsic rewards are designed to address challenges like sparse rewards and inefficient exploration. Below are the scaling coefficients for each algorithm, followed by the formal definitions. Additonally, these techniques use policies incorporating LSTMs to address partial observability:

- **IMPALA:** No intrinsic reward ($r_i = 0.0$).
- **COUNT:** $r_i = 0.005$.
- **RIDE:** $r_i = 0.1$.
- **CURIOSITY:** $r_i = 0.1$.
- **RND:** $r_i = 0.1$.
- **AMIGo:** $r_i = 0.1$ (applies to the teacher's intrinsic rewards).

The formal definitions of the intrinsic rewards are:

**COUNT:** The intrinsic reward is based on state visitation counts, encouraging exploration of less-visited states:

$$r_i = \frac{1}{N(s_0)},$$

where $N(s_0)$ is the (pseudo)count of visits to state $s_0$. Counts are never reset during training.

**RIDE (Rewarding Impact-Driven Exploration):** The intrinsic reward combines state novelty and state-change impact:

$$r_i = \|\phi(s) - \phi(s_0)\|_2 \cdot \frac{1}{N(s_0)},$$

where $\phi$ is trained to minimize both forward and inverse dynamics prediction errors. Counts $N(s_0)$ are reset at the beginning of each episode.

**CURIOSITY:** The intrinsic reward comes from the prediction error of a forward dynamics model $f$, which predicts the next state embedding $\phi(s_0)$ from the current embedding $\phi(s)$ and action $a$:

$$r_i = \|f(\phi(s), a) - \phi(s_0)\|_2.$$

**RND (Random Network Distillation):** The intrinsic reward is computed as the prediction error of a trainable network $\phi$ attempting to match the output of a fixed random network $\hat{\phi}$:

$$r_i = \|\phi(s_0) - \hat{\phi}(s_0)\|_2.$$

**AMIGo:** The teacher policy generates goals $g$ for the agent, with rewards given as:

$$r_i = v(s_t, g) = \begin{cases} +1 & \text{if } s_t \text{ satisfies } g, \\ 0 & \text{otherwise.} \end{cases}$$

The total reward is a weighted sum of intrinsic and extrinsic rewards:

$$r_t = \beta r_i + \alpha r_e, \quad \text{with } \beta = 0.3, \alpha = 0.7.$$

#### D.2.3 Algorithm-Specific Hyperparameters and Architectures

- **IMPALA (Baseline):**
  - Intrinsic reward: None.
  - Loss: Policy gradient, baseline, and entropy losses.
- **COUNT:**
  - Intrinsic reward: State visitation counts.
  - Uses count reset probability: $p = 0.001$.
- **CURIOSITY:**
  - Intrinsic reward: Forward prediction error.
  - Modules:
    - State embedding model: Encodes observations into 256-dimensional embeddings.
    - Forward dynamics model: Predicts next state embedding given current embedding and action.
    - Inverse dynamics model: Predicts action given two successive state embeddings.
  - Loss weights:
    - Forward dynamics loss: $10.0$.
    - Inverse dynamics loss: $0.1$.
- **RIDE:**
  - Intrinsic reward: Product of state visitation counts and the norm of state embedding changes.
  - Modules: Same as CURIOSITY.
- **RND:**
  - Intrinsic reward: Prediction error between random target network and predictor network embeddings.
  - Modules:
    - Random target network: Produces fixed embeddings for observations.
    - Predictor network: Trained to predict random target embeddings.
  - Loss weight: $0.1$.
- **AMIGo:**
  - Intrinsic reward: Teacher-generated rewards.
  - Teacher-specific parameters:
    - Intrinsic reward coefficient ($\beta$): $0.3$.
    - Extrinsic reward coefficient ($\alpha$): $0.7$.
    - Generator batch size: $150$.
    - Generator entropy cost: $0.05$.
    - Generator threshold: $-0.5$.

# APPENDIX E EXAMPLE ENVIRONMENT WITH LEARNED RULES

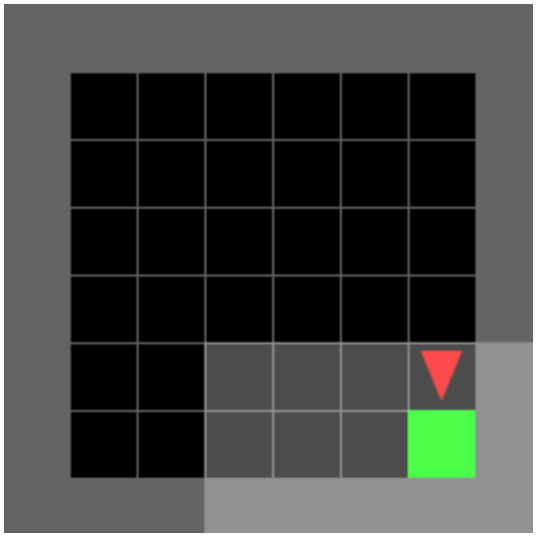


Figure 8: Illustration of Minigrid-Empty-8x8

The following are all the rules NACE learns in the Minigrid-Empty-8x8 environment as illustrated in Fig. 8:

```
Agent interacting with goal location:
<(vars=[1], shape[ 0, 0 ]='x', shape[ 0, 1 ]='H', ^down ) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.
<(vars=[1], shape[-1, 0 ]='H', shape[ 0, 0 ]='x', ^left ) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.
<(vars=[1], shape[ 0,-1 ]='H', shape[ 0, 0 ]='x', ^up   ) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.
<(vars=[1], shape[ 0, 0 ]='x', shape[ 1, 0 ]='H', ^right) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.

Goal location interacting with agent:
<(vars=[1], shape[ 0,-1 ]='x', shape[ 0, 0 ]='H', ^down ) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.
<(vars=[1], shape[ 0, 0 ]='H', shape[ 1, 0 ]='x', ^left ) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.
<(vars=[1], shape[ 0, 0 ]='H', shape[ 0, 1 ]='x', ^up   ) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.
<(vars=[1], shape[-1, 0 ]='x', shape[ 0, 0 ]='H', ^right) =/> (vars=[0], shape[ 0, 0 ]='.', score+=1)>.

Agent interacting with empty space:
<(vars=[1], shape[ 0, 0 ]=' ', shape[ 0, 1 ]='x', ^up   ) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.
<(vars=[1], shape[-1, 0 ]='x', shape[ 0, 0 ]=' ', ^right) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.
<(vars=[1], shape[ 0,-1 ]='x', shape[ 0, 0 ]=' ', ^down ) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.
<(vars=[1], shape[ 0, 0 ]=' ', shape[ 1, 0 ]='x', ^left ) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.

Empty space interacting with agent:
<(vars=[1], shape[ 0,-1 ]=' ', shape[ 0, 0 ]='x', ^up   ) =/> (vars=[1], shape[ 0, 0 ]=' ', score+=0)>.
<(vars=[1], shape[ 0, 0 ]='x', shape[ 1, 0 ]=' ', ^right) =/> (vars=[1], shape[ 0, 0 ]=' ', score+=0)>.
<(vars=[1], shape[ 0, 0 ]='x', shape[ 0, 1 ]=' ', ^down ) =/> (vars=[1], shape[ 0, 0 ]=' ', score+=0)>.
<(vars=[1], shape[-1, 0 ]=' ', shape[ 0, 0 ]='x', ^left ) =/> (vars=[1], shape[ 0, 0 ]=' ', score+=0)>.

Agent interacting with wall:
<(vars=[1], shape[ 0,-1 ]='o', shape[ 0, 0 ]='x', ^up   ) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.
<(vars=[1], shape[ 0, 0 ]='x', shape[ 1, 0 ]='o', ^right) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.
<(vars=[1], shape[ 0, 0 ]='x', shape[ 0, 1 ]='o', ^down ) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.
<(vars=[1], shape[-1, 0 ]='o', shape[ 0, 0 ]='x', ^left ) =/> (vars=[1], shape[ 0, 0 ]='x', score+=0)>.

Wall interacting with agent:
<(vars=[1], shape[ 0, 0 ]='o', shape[ 1, 0 ]='x', ^left ) =/> (vars=[1], shape[ 0, 0 ]='o', score+=0)>.
<(vars=[1], shape[ 0, 0 ]='o', shape[ 0, 1 ]='x', ^up   ) =/> (vars=[1], shape[ 0, 0 ]='o', score+=0)>.
<(vars=[1], shape[-1, 0 ]='x', shape[ 0, 0 ]='o', ^right) =/> (vars=[1], shape[ 0, 0 ]='o', score+=0)>.
<(vars=[1], shape[ 0,-1 ]='x', shape[ 0, 0 ]='o', ^down ) =/> (vars=[1], shape[ 0, 0 ]='o', score+=0)>.
```