Surgical Interventions for Causal Exploration with LLM-Based Agents

by

Arthur Mercier

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday November 26, 2025 at 8:45.

Student number:	5056195	
Project duration:	November 22, 2024 – November 26, 2025	
Thesis committee:	Mr. Chirag Raman,	TU Delft, Daily supervisor
	Mr. Marcel Reinders,	TU Delft, Advisor
	Mr. Avishek Anand,	TU Delft, External committee member
	Mr. Ojas Shirekar,	TU Delft, Daily Co-Supervisor

Surgical Interventions for Causal Exploration with LLM-Based Agents

Arthur Mercier

Abstract

This paper explores whether explicit causal reasoning can enhance exploration in embodied LLM-driven agents by integrating a causal world model (BISCUIT) and a novel surgical intervention mechanism. We introduce two new agent architectures: Predforge, which uses causal predictions to inform action selection, and Causalforge, which identifies and executes surgical interventions to isolate causal dependencies. We develop a full evaluation pipeline including an exploration metric, intervention detection framework, and multi-agent experimental setup in AI2-THOR and compare these agents against Voyager and Mindforge baselines. Our results show that BISCUIT's prediction errors are concentrated precisely in the semantically important regions of the environment, limiting Causalforge's ability to identify most surgical interventions. However, the predictions remain sufficiently reliable to provide modest benefit to Predforge. Multi-agent experiments further reveal how communication, partner modeling, and environment structure shape exploration. We conclude with a detailed analysis of failure modes and outline future directions including online causal world model updates, integrating Mindforge beliefs into the causal world model, richer causal environments, and task independent skill acquisition to unlock the full potential of causal exploration in LLM-based agents.

1 Introduction

Large Language Model (LLM)-based agents have shown strong potential in navigating embodied environments[1][2]. By leveraging the extensive knowledge encoded during their training, they are able to effectively generate plans and execute actions autonomously[3]. This large context received during training also makes them out-of-the-box generalizable to environments through small in context changes without requiring extensive retraining. A key advantage of LLM-based agents is their ability to communicate in natural language, enabling effective collaboration with other agents and even humans in multi-agent tasks. This is especially useful for tasks that require combined planning or for heterogeneous setups where each agent possess different capabilities. Furthermore, LLM agents have also been endowed with the ability to model their partner agents to improve coordination and collaboration [4]. The BigTom template suggested by Gandhi et al. [5] allows an agent to infer a partners perceptions, beliefs, desires and actions, enhancing its ability to reason about and plan in response to its partner's behavior.

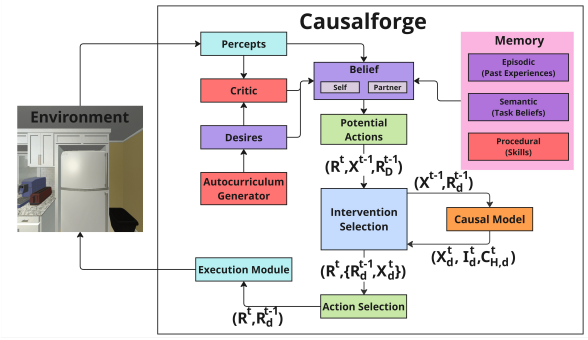

Figure 1: Causalforge agent's architecture, illustrating how the causal world model integrates with our novel surgical action selection module.

Despite these strengths, LLM-based agents face a significant challenge in its lack of consistency when engaging with the causal aspects of the environment. Because LLMs operate based on statistical models, they often make spurious correlations based on patterns derived from past experience and training data [6], instead of relying on causal links that would help it reason about changes in the environment[7]. These links appear whenever the state of a variable in the environment deterministically depends on previous states. Lippe et al. [8] has already proposed a framework for integrating a causal model with an LLM for visual embodied environments. By using Causal Representation Learning (CRL), the agent can model the causal links of the environment and inform the LLM-agent about the effects of potential actions, improving decision making and helping agents build a better causal understanding of their environment.

For an agent to efficiently learn causal relationships in its environment, it must perform surgical interventions [9] that will allow it to disentangle the environment's causal variables from each other. Surgical interventions are targeted actions that isolate causal variables from each other to determine their effects on one another. These actions can be visualized as pushing the first domino in a line or placing a barrier between two dominoes to test their dependence. The agents ability to find and perform these surgical interventions will determine how fast

it is able to learn the causal relationships within the environment.

Causal learning becomes even more complex in multi-agent environments, where other agents' actions can interfere with the causal links an LLM-based agent is trying to learn. This is especially problematic for deep learning which has a hard time adapting to changes in distribution as it tends to overfit on the training distribution[7]. However, by modeling partner actions as causally dependent on their beliefs [5], their behavior can be incorporated into the causal model as another process in the environment, allowing the agent to reason jointly about environmental dynamics and its interactions with other agents. An agent that has learned causal links can use that knowledge to better interact with and explore other environments in similar situations. This would create an agent more robust to changes within the environment possibly resulting from another actor.

This raises a central research question: **How can LLM-agents leverage causal learning to explore multi-agent environment more efficiently?**

To address this question, we propose:

1. A platform and metric for evaluating LLM-based agents, such as Voyager and Mindforge, on their ability to explore simulated environments.
2. An intrinsic motivation mechanism that encourages agents to perform surgical actions for more efficient causal discovery.
3. An experimental evaluation of our approach in a causal environment built in AI2-THOR [10], assessing the agent's ability to learn causal links, predict action effects and interact with the world.

The remainder of this paper is structured as follows: Chapter 2 summarizes the related literature, while Chapter 3 outlines the technical foundations drawn from prior work used in this contribution. Chapter 4 defines our problem which is then approached in Chapter 5, detailing the architecture of our agent, causal model, and our experimental setup. Chapter 6 showcases and discusses our results. Finally, Chapter 7 concludes with insights and potential directions for future research.

2 Previous work

2.1 Large Language Model Agents

Voyager autonomous agent. Wang et al. [3] leveraged the autonomous capabilities of large language models (LLMs) to create Voyager, a lifelong learning agent designed for open-ended environments like Minecraft, where players or agents must explore, gather resources, and gradually advance through a tech tree. They argued that a successful lifelong agent should mirror human learning by navigating its world based on current skills, proposing its own tasks, learning from experience, and maintaining intrinsic motivation to explore. Voyager integrated an LLM into an embodied agent through three modules: an automatic curriculum, a skill library, and an iterative prompting mechanism. The automatic curriculum decomposes complex goals into achievable subtasks using chain-of-thought reasoning and encourages exploration by generating new, unique challenges for the agent to complete. The skill library stores learned behaviors as interpretable, reusable programs refined through iterative prompting. After each execution, a Critic LLM evaluates whether the task was successfully achieved. If that is the case, the resulting skill is stored for future use. Otherwise, the Critic provides feedback for improvement or triggers a new task after repeated failures. Together, these modules enable Voyager to continuously learn and generalize beyond imitation or reinforcement-based methods, allowing it to outperform state-of-the-art agents by collecting more diverse items, traveling further, and progressing higher in Minecraft's tech tree.

Mindforge social learning. Lică et al. [4] identified a limitation in Voyager, noting that its LLM (ChatGPT 3.5) had prior knowledge of Minecraft from the game's Wiki, which biased its performance, whereas open-weight models like Mistral failed to perform basic tasks. To overcome this, they developed Mindforge, an extension of the Voyager architecture inspired by the Social Intelligence Hypothesis [11], where an expert agent (ChatGPT) teaches an open-weight LLM to complete tasks collaboratively. Mindforge introduces three additional modules: a causal Theory of Mind (ToM) template [5] for modeling other agents, a natural language communication interface, and a Soar cognitive architecture [12] for memory and reasoning. Agents operate under a Belief-Desire-Intention (BDI) framework [13], where beliefs represent environmental understanding (including models of partner agents), desires are goals from the autocurriculum, and intentions are action plans generated by the LLM. The Soar memory consists of episodic (past interactions), semantic (general facts), and procedural (learned skills) components. Using natural language dialogue, weaker agents can seek help from stronger ones, updating their beliefs and correcting misunderstandings or code errors. Through this collaborative reasoning and communication, Mindforge resolves Voyager's failure cases, such as false beliefs or non-executable code and significantly outperforms it in both tech tree progression and diversity of items collected, demonstrating that open-weight LLMs can achieve strong performance through interaction with expert agents.

2.2 Causality

Intuitive theories and counterfactual dependencies. Humans learn by understanding intuitive theories, which Gerstenberg and Tenenbaum [14] describe as structured systems of causal laws that explain how concepts interrelate. These intuitive theories act like probabilistic simulators, generating a probability distribution over possible outcomes rather than a single deterministic result, allowing humans to reason about uncertainty, make predictions, and infer hidden causes. Within the dependency theory framework [15, 16], counterfactual dependencies capture "what if" reasoning, examining how an effect would change if its cause were altered or removed. By combining probabilistic and counterfactual dependencies, as proposed by Gerstenberg and Tenenbaum [14], causality can be modeled as shifts in these probability distributions, enabling both humans and agents to perform prediction, inference, explanation, and planning in a manner similar to human cognitive learning.

Types of causal machine learning. Schölkopf et al. [17] propose Causal Representation Learning (CRL) as a way to model the causal structure of an environment directly. Building on this idea, Komanduri et al. [6] categorize causal models into two main scenarios: CRL and Controllable Counterfactual Generation (CCG). CCG focuses on learning mappings from known causal variables, available as labeled data, to observed outcomes. CRL requires discovering both the causal variables and their underlying structure from observational, interventional, or counterfactual data, corresponding to Pearl's Causal Hierarchy of "seeing", "doing", and "imagining". CRL methods are further divided by the nature of the environment: temporal environments, where states evolve over time, and static ones, where causal relations remain fixed. BISCUIT [18], a causal world model (CWM), learns causal links in temporal environments using interventional data by encoding raw observations into latent causal variables, modeling their dependencies across time, and decoding predictions back into observations. By iteratively predicting next states and minimizing prediction errors, BISCUIT uncovers the true causal structure of the environment and the influence of the agent's actions. Extending this approach, Lippe et al. [8] integrate BISCUIT with an LLM-based agent: the CWM processes visual and linguistic embeddings of the environment and actions, predicts likely outcomes, translates these back into natural language, and feeds them to the agent. This feedback loop enables the agent to anticipate and reason about the effects of its actions, improving decision-making and planning.

Surgical Interventions for causal disentanglement. For the agent to effectively navigate the environment the CWM needs to be able to learn these causal links efficiently. Surgical interventions [9] are actions that block causal variables from interacting with each other. This enables the agent to examine the effect of a variable without confounding on the other. Cohen [9] defines a causal link as a function of the state of the environment and the intervening action:

$$\text{outcome}_Y^a = \text{proc}_Y \ \text{do}_X(a) : X \to Y \qquad (1)$$

where $\text{do}_X(a) : X \to X$ changes the state of the environment X based on the action performed a and $\text{proc}_Y : X \to Y$ is a function that returns the causal outcome Y of a state X. It is thus assumed that the causal mechanism that determines the outcome deterministically depends on the state X and the intervention a.

Further decomposing Y into Y_i each containing a subset of outcome variables enables for comparisons to be made between them. Y_I and Y_J are said to have a **relation** if Y_I can inform us on the possible values (or values to rule out) of Y_J. If for each possible outcome of Y_I there exists only one possible outcome of Y_J we call it a **functional relation**, $f^a : Y_I \to Y_J$. outcome_J^a is **determined** by outcome_I^a via f^a. Cohen [9] defines a good surgical intervention as an action that replaces the mechanism for Y_i without changing the mechanisms for other variables. By performing these surgical actions an agent can efficiently disentangle the causal links within the environment enabling the CWM to learn them faster.

3 Background

3.1 Agent Modules

We have re-implemented several modules from the Voyager [3] and Mindforge [4] agent architectures. The auto curriculum generator, critic and skill memory system stem from Voyager, whereas the communication module, belief system, episodic and semantic memory are taken from Mindforge.

Automatic curriculum generator The automatic curriculum module breaks down complex tasks into simpler, achievable goals based on the agent's state, a history of previously attempted tasks (succeeded or failed) and directives to encourage the agent to explore. These simpler tasks are generated using 'chain of thought' which enables the LLM to reason and iterate on a task until it is deemed feasible to be performable in the current environment. The automatic curriculum will keep pushing the agent to find new unique items and skills it could learn, endowing the agent with an intrinsic motivation to explore its surroundings and find as many possible interactions with the environment as it can.

Skill memory and critic. The Skill library stores skills as a key-value pair in a vector database

where the key is a description of the program and the value is program code. Before storing for reuse, the program is first revised to make it more interpretable and generalizable. This allows the agent to use the program code as a template for performing future more complex skills. When refining the program for storage, the LLM is given as input: guidelines for code generation, API's and learned skills relevant to the situation, the agents current state, the program performed in the previous round and what was observed as a result. After going through an iterative prompting mechanism to refine the program it is stored in the skills library where it can then be retrieved by indexing on the embedding of self-generated task plans and an observation of the environment. At each round of the iterative prompting mechanism, the program is run in the environment where observations of the environment and any execution errors are then made. These are then passed to an independent LLM which acts as a critic and decides if the task was completed or failed. The critic also provides suggestions on how to complete the task. If this fails for 4 rounds, a new task is generated by the automatic curriculum instead. These additions to the agent enabled Voyager to outperform other state of the art agents at exploring the Minecraft environment. Voyager was able to find more unique items, cover more ground and climb up the tech tree faster and further than other agents.

In Mindforge, agents are modeled based on their Beliefs, desires and actions. This fits into the Belief-Desire-Intention framework proposed by Rao and Georgeff [13]. Desires represent the goals generated by the autocurriculum generator and intentions refer to the action code generated by the LLM. Belief encapsulates what the agent thinks the current state of the environment is based on what it has learned through its interaction with the environment and communication with other agents. Belief also includes other partner agents in the environment which the agent models with a recursive BDI framework. Thus for every partner agent in the environment, the current agent holds a separate BDI representation within their belief. At each timestep, an agents belief depends on the observation received from the environment and relevant past experiences fetched from the Soar memory. The Soar architecture is divided into three separate databases: episodic, semantic and procedural memory.

Episodic and semantic memory. Firstly, the episodic memory stores past interactions and their outcomes. This can help it reason about how new interactions can be adapted for a better chance of success. Secondly, the semantic memory stores facts distilled from many distinct interactions. These are facts that don't have a direct connection with a certain episode but hold true throughout the whole environment such as the necessary requirements to achieve a certain task. Lastly, the procedural memory stores the skills the agent has learned just as in Voyager. These skills are stored as code that can directly be called upon to preform the associated actions without the agent having to relearn them.

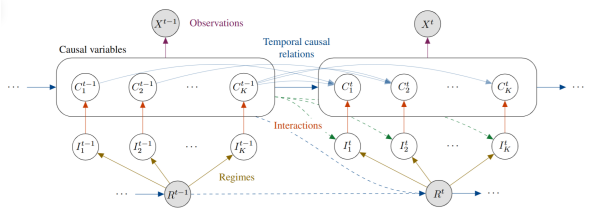

Figure 2: Visualization of the causal world model put forwards by BISCUIT [18]

3.2 BISCUIT

BISCUIT describes the state of the environment by K latent causal variables which interact with each other over time. Each causal variable $\mathcal{C}^t = \{C_1^t, \ldots, C_k^t\}$ depends on a subset of variables in the previous timestep $\mathcal{C}^{t-1} = \{C_1^{t-1}, \ldots, C_k^{t-1}\}$ (This can be trivially extended to longer dependencies [18]). The structure of the interactions between the causal variables is assumed to be time invariant. At each timestep t, an agent may interact with the environment by performing an action R^t. For each causal variable C_i^t, a latent binary interaction variable I_i^t describes if the causal variable is affected by the action R^t. I_i^t is considered to be a function of the current action and the last environment state $I_i^t = f_i(R^t, C^{t-1})$. X^t is the observation received by the agent after acting in the environment. g is the injective observation function that maps the causal variables to the observation $X^t = g(C^t)$. This means each possible value of $\mathcal{C}^t$ maps to a distinct and unique observation. BISCUIT assume that there exists no confounding variables other than those mentioned as part of the causal world model. Using this causal world model, once the causal variables are discovered, the full causal graph can be learned by testing for conditional independence between $\mathcal{C}^{t-1}$ and $\mathcal{C}^t$. Figure 2 shows how the causal variables are affected by the agent's action and the previous timestep.

A causal world model can be described by $M =< g, f, \omega, \mathcal{C} >$ where ω parameterises the latent distribution:

$$p_\omega(\mathcal{C}^t|\mathcal{C}^{t-1}, R^t) = \prod_{i=1}^{K} p_{\omega,i}(C_i^t|\mathcal{C}^{t-1}, f_i(R^t, \mathcal{C}^{t-1})) \tag{2}$$

The goal of the method proposed by BISCUIT [18] is to learn a model $\hat{M} =< \hat{g}, \hat{f}, \hat{\omega}, \hat{\mathcal{C}} >$ that estimates the true causal world model $M =< g, f, \omega, \mathcal{C} >$ by using the previous observation X^{t-1} and action R^t to try to predict what the next observation X^t will

be after the action is performed and all causal systems resolve. Substituting this in equation 2 gives us the prediction function for the causal model:

$$p_{\text{CWM}}(\hat{X}^t|X^{t-1}, R^t) = g(p_\omega(\mathcal{Z}^t|g^{-1}(X^{t-1}), R^t)) \tag{3}$$

where $\mathcal{Z}^t$ is the causal model's weights, trying to approximate the true environment variable $\mathcal{C}^t$.

4 Problem Statement

Let us denote the input to the agent as $X^t = \{\mathbf{i}^t, j^t\} \forall t \in [T]$ where $\mathbf{i}^t \in \mathbb{R}^{c \times h \times w}$. We want the agent to decide on the next action $(R^t, M^t) = \text{agent}(X^{t-1})$, where $R^t \in \mathcal{A}$ is an action an agent can take in a given environment $\mathbb{E}$ and $M^t \in \mathcal{M}$, here $\mathcal{A}$ is the set of all actions and $\mathcal{M}$ is the power set of all tokens the agent can choose to output.

We now introduce a Causal World Model $\hat{(}X^{t+1}, \mathcal{C}_H^t) = \text{CWM}(X^t, R^t)$ where $\hat{X}^{t+1}$ is the predicted observation that would be received if R^t was performed in the environment and $H \subset \mathcal{C}^t$ is the set of latent causal variables that action R^t has interacted with.

In multi-agent settings the environment is also affected by partner agent's actions $(R_a^t, M_a^t) = \text{partner}(X^{t-1})$. By modeling the partner agent as part of the environment causal process, $(\hat{X}^{t+1}, \mathcal{C}_H^t) = \text{CWM}(X^t, R^t)$ holds.

5 Methodology

5.1 Agent architecture

In addition to Voyager [3] and Mindforge [4] we introduce two new autonomous LLM agents, **Predforge** and **Causalforge**. **Predforge** extends Mindforge by incorporating a CWM that provides it with a prediction of the environment state $\hat{X}^t$ based on the current environment X^{t-1} and an envisioned action R^t. Rather than executing the Mindforge agent's action directly in the environment, the action is first passed through the CWM, which predicts its potential effect on the environment. This predicted outcome is then returned to the agent, enabling it to make a more informed decision about which action to actually perform.

Causalforge builds upon this concept by identifying possible surgical interventions available to the agent through a two-step prediction process. Figure 1 illustrates all the modules in Causalfroge and their interactions with each other and with the CWM. The surgical action selection module receives the current environmental observation X^{t-1}, the envisioned action R^t and a set of candidate interventions $\mathcal{R}_D^t$ that the agent is considering. The CWM module then returns to the agent prediction pairs $\{R_d^t, \hat{X}_d^t\}$ $\forall R_d^t \in \mathcal{R}_D^t$ where $\hat{X}_d^t = \text{CWM}(\text{CWM}(X^{t-1}, R_d^t), R^t)$. Using these prediction pairs, the agent can determine which pair of actions constitute surgical interventions and decide on which one to execute in the environment. The core assumption being that providing the agent with knowledge about the causal effects of its actions enhances its planning ability and thereby improves its exploration capacity.

```
success, critique = Critique(X^{t-1}, R^{t-1}, task,
↪  task_belief, communication, error)

if success:
    save_skill(R^{t-1}, task_belief)
    task, task_beliefs = Auto_curriculum(X^{t-1},
    ↪  R^{t-1}, communication, success, critique)
save_episode(task, task_belief, R^{t-1}, critique,
↪  success)

skills = Skill_memory(task, X^{t-1})
episodes = Episodic_memory(task, X^{t-1})

# beliefs
perception_beliefs =
↪  Create_perception_beliefs(X^{t-1}, communication,
↪  error)
partner_beliefs =
↪  Update_partner_beliefs(partner_beliefs,
↪  communication)
interaction_beliefs =
↪  Update_interaction_beliefs(interaction_beliefs,
↪  task, communication)
task_beliefs = Update_task_beliefs(task_beliefs,
↪  task, interaction_beliefs)
beliefs = {perception_beliefs, partner_beliefs,
↪  interaction_beliefs, task_beliefs}

R^t, outgoing_communication = Action_selection(
    X^{t-1}, R^{t-1}, task, critique, error, skills,
    ↪  episodes, beliefs
)

R_D^t = Candidate_action_selection(R^t,  X^{t-1}, R^{t-1},
↪  task, critique, error, skills, episodes,
↪  beliefs)

{R_d^t, X̂_d^t} = Surgical_intervention_selection(R^t,
↪  R_D^t, X^{t-1})
```

Figure 3: Main-Loop Pseudoscope

Figure 3 describes how a typical iteration unfolds. The process begins with the critique module, which evaluates whether the previous action successfully completed the assigned task. If the task was successful the success flag is set to true, otherwise it is set to false, and a critique is generated containing suggestions on what may have failed and how to improve in the next attempt. When a task is successfully completed, the past action is saved as a skill and the auto-curriculum generates both a new task and initial beliefs about how that task might be accomplished. New tasks are generated based on the agent's currently known skills, as well as its past successes and failures. The objective of the auto-curriculum is to produce a task that the agent can reasonably accomplish given its current capabilities and the state of the environment around it. Regardless of success or failure, the previous episode is stored in memory.

With our current task established, newly generated or from the last iteration, the agent retrieves all the relevant skills and episodes from memory that may assist in choosing its next action. The agent then forms its perceptual beliefs about the environment, updates its beliefs about its partner, and integrates any new beliefs created during the interaction with partner agents. These interaction beliefs are then used to refine the agent's task beliefs, shaping its understanding of how the task could be completed. This belief-updating process allows agents to incorporate communicated advice from one another about potential task solutions, filtered through their beliefs about the other agent's competence.

Once all relevant information is gathered, it is passed to the action selection module which determines the next action to attempt in the environment. The main distinction between Voyager and Mindforge is that Voyager's action selection module only receives information from the auto-curriculum, critic and skill memory. Mindforge, in contrast, also integrates episodic memory and beliefs. For both Voyager and Mindforge, the selected action is executed directly in the environment.

Predforge, however, will first use its CWM to simulate the outcome of that action in the current environment. The action selection module is then called again, this time with the added information of the envisioned action and its predicted outcome. Based on this new information, the agent decides whether to proceed with the action or choose an alternative. The final action selected by Predforge is then performed in the environment.

Finally, Causalforge extends this process by examining the envisioned action selected in the last step of Mindforge to identify any potential surgical interventions available to the agent. If a surgical intervention exists for the current step and envisioned action, Causalforge performs it in the environment. In our setup, only one surgical intervention can exist in the environment at any given time, thus selecting the first available intervention is sufficient.

5.2 Causal model

For a causal model to predict the effects of an agent's actions in a temporal environment, it must be capable of learning from both observational and interventional data. We have identified four such candidate causal model [6], illustrated by figure 16 in the appendix. CITRIS and iCITRIS are precursors upon which BISCUIT built on. Lachapelle and Lacoste-Julien [19], on the other hand, propose a model structurally similar to CITRIS and BISCUIT but augmented with sparsity constraints, based on the assumption that causal links between agent actions and latent causal variables across time-steps are inherently sparse. In our setting, however, the agent interacts with other social agents, whose actions are influenced by beliefs, which are in turn influenced by numerous contextual factors. Under these conditions, enforcing a sparsity constraint would not be beneficial. Consequently, we selected BISCUIT as the causal model for our framework.

BISCUIT makes four primary assumptions about the environment in order to guarantee the identifiability of causal variables. The first assumption states that interactions between the agent and causal latent variables are binary in nature. Each action either interacts with a given causal variable or has no effect on it. Formally, every action R^t is associated with a set of binary indicators I^t, where each $I_i^t \in I^t$ designates whether action R^t interacts with causal variables $C_i^t \in C^t$.

The second assumption is that all causal links exhibit distinct interaction patterns. Causal latent variables are disentangled by observing the effect of interacting with specific parts of the environment, meaning two variables are not always interacted with at the same time. If two variables share the same interaction pattern and consistently produce identical effects, they are modeled as a single variable. This is not a problem for our case as it does not reduce the precision of modeling action effect predictions.

The third assumption states that causal relations unfold over time rather than within a single timestep. BISCUIT does not take in account causal links between variables in the same timestep, dependence is said to flow from one timestep (or action) to variables in future time-steps.

Finally, BISCUIT assumes that causal mechanisms vary sufficiently either through different interactions or across time. This variability is necessary for the model to correctly infer the underlying causal structure of the environment.

BISCUIT was originally designed to model the effects of a single agent's action on an environment governed by latent causal variables. In multi-agent environments however, the state of the next timestep depends not only on the agent's own actions but also on those of other agents. The Theory of Mind causal template proposed by Gandhi et al. [5] suggests that an agent's choice of action depends on both their belief and desire and that these beliefs are themselves dependent on the agent's perceptions. By decomposing these categories into B distinct latent variables, each corresponding to an individual belief or desire, we can establish explicit causal links between the belief and desire variables and the agents intended action. In other words, Gandhi et al. [5] posits the existence of two functions $p_{\omega,r}$ and $p_{\omega,b}$, describing the causal dependencies from {previous beliefs, observation} to current beliefs and from {current beliefs, desire} to intended action:

$$R_a^t = p_{\omega,r}(p_{\omega,b}(\mathcal{C}^t_{a,\text{ belief}}|X_a^{t-1}, \mathcal{C}^{t-1}_{a,\text{ belief}}), \mathcal{C}^t_{a,\text{ desire}}) \tag{4}$$

Rather than modeling both causal links separately, we define a single function $p_{\omega,m}$ that captures the overall causal dependence from previous beliefs, observation, desire directly to the environmental causal variables affected by the partner agent's action. By incorporating the belief and desire latent variables into the set of environmental causal variables, our causal model can learn the causal link $p_{\omega,m}$ in the same manner as it learns other environmental causal links. This formulation allows us to factor the partner agent's actions R_a^t of equation 4 into the broader set of causal variables $\mathcal{C}^t$ defined in equation 2.

$$\begin{aligned} p_\omega(\mathcal{C}^t|\mathcal{C}^{t-1}, R^t) =& (\prod_{i=1}^{K} p_{\omega,i}(C_i^t|\mathcal{C}^{t-1}, f_i(R^t, \mathcal{C}^{t-1}))) \\ &\cdot (\prod_{i=1}^{A}\prod_{j=1}^{B} p_{\omega,i,j}(C_{i,j}^t|\mathcal{C}^{t-1}, f_{i,j}(R_a^t, \mathcal{C}^{t-1})))) \\ &\text{where } \mathcal{C}^t = \{C_1^t, \ldots, C_K^t\} \cup \{C_{K+1}^t, \ldots, C_{K+B}^t\} \end{aligned} \tag{5}$$

$\forall a \in A$, partner agents in the environment

Although the true beliefs and desires of partner agents are not directly observable, the CWM can approximate them as latent variables $\mathcal{Z}^t$, inferred from the environment's previous observation X^{t-1}, in the same way it approximates the environmental causal variables $\mathcal{C}^t$. By substituting these true beliefs and desires with their approximations, we can estimate the partner agent's next action as $(\hat{R}^t, \hat{M}^t)$ = partner(X^{t-1}).

This dependence between the partner's action and the previous observation can be represented within the CWM as an additional causal link. Modeling this dependency directly allows us to bypass the second term of equation 5, enabling the CWM to predict the next environmental state based solely on the previous state and our agent's action R^t.

$$\begin{aligned} &p_{\text{CWM}}(\hat{X}^t|X^{t-1}, R^t) = g(p_\omega(\mathcal{Z}^t|g^{-1}(X^{t-1}), R^t)) \\ &\text{where } \mathcal{Z}^t = \{Z_1^t, \ldots, Z_K^t\} \cup \{Z_{K+1}^t, \ldots, Z_{K+B}^t\} \end{aligned} \tag{6}$$

Modeling partner agents as a function of our agents' actions and the current environment state introduces several trade-offs. Most notably, it portrays partner agents as reactive, meaning it may fail to capture the behavior of proactive agents that act independently of our main agent's influence. In our relatively small environment, however, agents constantly interact with one another, which minimizes this limitation. In contrast, open-world environments such as Minecraft, used by Wang et al. [3], present a greater challenge, as agents may operate in entirely different regions of the environment with little or no overlap in their interactions.

Another important consideration is that the observation X^t available to our agent must be sufficiently similar to that of the partner agent for the CWM to accurately predict the partner's perceptual beliefs. This condition holds in our environment, where agents are located side by side and share the same visual field. In settings like Minecraft, however, partner agents may wander off for extended periods and interact with unseen portions of the environment, leading to beliefs that the CWM cannot reliably infer. Finally, we assume that agents do not possess any preset beliefs and desires unknown to the causal model. This assumption is valid in our setup, as agents are initialized without prior beliefs or desires and develop them solely from environmental observations X^t as input.

Our causal world model module outputs not only a prediction of the next state, but also a set of binary variables I^t, describing which causal variables $\mathcal{C}^t$ are affected by the agent's action R^t. Additionally, the CWM provides access to the prior distribution p_w from which these predictions are sampled. By examining changes in this prior across time-steps, particularly for those variables with which the agent has interacted $I_i^t = 1$, we can gain deeper insight into the causal effects of different actions on the environment.

5.3 Surgical action selection

Previous research has demonstrated significant success in equipping agents with intrinsic motivation functions to encourage specific beneficial behaviors [20][21][22]. One such function used in multi-agent settings, is the social intrinsic motivation (SIM) mechanism proposed by Jaques et al. [23]. SIM promotes collaboration by biasing the agent toward actions that maximize social impact. This is done through reasoning about potential future actions and their counterfactual effect on the behavior of other agents. The action that produces the greatest change in the behavior of other agents is deemed the most influential and is therefore selected.

While SIM has proven effective for fostering cooperation through improved communication and coordination, it is not well suited for exploring causal variables. Focusing exclusively on actions with maximum impact hinders the agent's ability to disentangle causal dependencies, as successful disentanglement requires observing the effects of interacting with one variable while leaving others unaffected. If the agent continuously selects actions that influence all variables simultaneously, it becomes impossible to determine which observed effects are the result of a dependence with which causal variables. A more effective exploration strategy would involve selecting actions that isolate small subsets of variables,

enabling the agent to observe their individual effects and thereby achieve more precise causal understanding.

Cohen [9] argues that **surgical interventions** provide an effective means of disentangling dependencies between causal variables by isolating interactions to specific subsets of those variables. A surgical intervention replaces an existing causal link $p_{\omega,i}$ with a new one, while leaving the causal links of other variables unaffected. This selective modification allows the resulting observation to directly reveal information about the dependencies among the causal variables that were targeted by the intervention.

In our model, an established causal link corresponds to the mapping between an agent's action and the subset of causal variables it influences, expressed as $R^t \rightarrow \mathcal{C}_H^t$ where $C_H^t \subset \mathcal{C}^t$. To better understand this relation, we can perform an intervention consisting of an additional, preceding action. Specifically, executing a sequence of actions $R^t R^{t-1} \rightarrow \mathcal{C}_H^t$, allows us to intentionally disrupt the original causal link, thereby altering the resulting values of the affected variables $\mathcal{C}_H^t$ while leaving the remaining variables $\mathcal{C}^t \backslash \mathcal{C}_H^t$ unchanged.

A proposed framework for selecting surgical interventions involves using our causal model to predict which subset of causal variables $\mathcal{C}_H^t$ would be affected by a given intervention R_d^t.

$$\mathcal{C}_H^t = \{C_d^t |\ p_\omega(C_d^t | \mathcal{C}^{t-1}, R^t) \neq C_d^{t-1}\} \tag{7}$$

The subset $\mathcal{C}_H^t$ can be directly derived from the binary interaction variables I^t provided by BISCUIT, which indicate which causal variables are influenced by a specific action. To determine whether an intervention has effectively altered the environment, we compare the values of the affected variables C_I^t in the prior distribution both with and without the intervention.

Using this information, we can identify which preceding actions R_d^{t-1} within the set of possible actions $\mathcal{R}_D^{t-1}$ produce changes in the values of certain variables in $\mathcal{C}_H^t$ without affecting any of the remaining variables $\mathcal{C}^t \backslash \mathcal{C}_H^t$. Actions that satisfy these criteria can thus be classified as surgical interventions:

$$\begin{gathered} R_d^{t-1} \in \mathcal{R}_D^{t-1} \\ \text{where } p_\omega(\mathcal{C}_H^t \mid \mathcal{C}^{t-1}, R^t) \neq p_\omega(\mathcal{C}_H^t \mid \mathcal{C}^{t-1}, R_d^{t-1} R^t) \\ p_\omega(\mathcal{C}^t \setminus \mathcal{C}_H^t \mid \mathcal{C}^{t-1}, R^t) = p_\omega(\mathcal{C}^t \setminus \mathcal{C}_H^t \mid \mathcal{C}^{t-1}, R_d^{t-1} R^t) \end{gathered} \tag{8}$$

To able the agent to identify surgical interventions during exploration, we introduce a surgical selection module into the Mindforge architecture. This module is positioned at the end of the Mindforge loop (see Figure 3), once the agent has selected an action to perform in the environment. The module receives three inputs: the current observation X^{t-1}, an action R^t and a set of candidate actions $\mathcal{R}_D^{t-1}$. Here, R^t represents the action the agent intends to execute in the environment, generated using the same architecture as Mindforge. The set of candidate actions $\mathcal{R}_D^{t-1}$ can be interpreted as the set of all possible actions, though a language model is used to filter out actions irrelevant to the part of the environment affected by R^t, thereby reducing computational load.

First, the module uses the CWM to predict the effect of the chosen action R^t on the current environment:

$$\hat{X}^t,\ \hat{I}^t,\ \hat{\mathcal{C}}_H^t = \text{CWM}(R^t,\ X^{t-1}) \tag{9}$$

Here, $\hat{X}^t$ denotes the predicted observation after performing action R^t, $\hat{I}^t$ represents the indices of the causal variables affected by the action and $\hat{\mathcal{C}}^t$ corresponds to the prior distribution from which the predicted observation $\hat{X}^t$ is generated.

Next, for each candidate action $R_d^{t-1} \in \mathcal{R}_D^{t-1}$, a two-step prediction process is performed. First, the effect of candidate intervention R_d^{t-1} is predicted. Then, this predicted observation is used as input for a second prediction simulating the effect of executing R^t after candidate intervention R_d^{t-1}:

$$\begin{gathered} \hat{X}_d^{t-1},\ \hat{I}_d^{t-1},\ \hat{\mathcal{C}}_{H,d}^{t-1} = \text{CWM}(R_d^{t-1},\ X^{t-1}) \\ \forall\ R_d^{t-1} \in \mathcal{R}_D^{t-1} \end{gathered} \tag{10}$$

$$\hat{X}_d^t,\ \hat{I}_d^t,\ \hat{\mathcal{C}}_{H,d}^t = \text{CWM}(R^t,\ \hat{X}_d^{t-1}) \ \forall\ \hat{X}_d^{t-1} \tag{11}$$

The next step involves expanding the set of interactional variables. Although BISCUIT disentangles the environment into multiple latent causal variables, multiple variables may correspond to different aspects of the same physical object. For instance, a microwave is represented by two causal variables. One variable indicates its open/closed state and the another represents whether it is turned on or off. However, the interactional variables only include those directly linked to the performed action. An action that opens the microwave door interacts only with the open/closed variable and not the on/off one.

To address this, we expand the interactional variables to include all causal variables associated with the state of the object being interacted with. BISCUIT provides a dictionary mapping each environmental object to its relevant set of latent variables. This dictionary is constructed during training by grouping together variables whose values strongly influence the final state of the same environmental element. While the labeling of these groups is done manually, the labels themselves are not required for

the variable expansion process. This expansion is applied to the interactional variables of both the original action $\hat{I}^t$ and each candidate action $\hat{I}_d^{t-1}$.

With these expanded subsets, we can determine whether the interactional variables of the resulting intervention $\hat{I}_d^t$ form a subset of the union of both expanded sets $\hat{I}^t \cup \hat{I}_d^{t-1}$. If this condition holds, the intervention can be considered surgical, as it interacts only with the intended portion of the environment. If not, the intervention is deemed non-surgical, indicating that additional, unintended parts of the environment were affected.

The final step in identifying valid surgical interventions involves confirming that the resulting environment state was indeed influenced by the intervention. This is achieved by comparing the prior distribution of the original action $\hat{\mathcal{C}}^t$ to the prior obtained after the two-step prediction with the candidate intervention $\hat{\mathcal{C}}_d^t$. Only the causal variables that were interacted with are compared, ensuring that unrelated environmental changes do not affect the evaluation. Furthermore, causal variables that change consistently across all candidate actions are excluded, as they likely represent random factors or slow-changing variables unrelated to the current intervention. During initialization, BISCUIT creates more latent variables than there actually are in the environment to ensure full coverage of potential causal factors. The random variables that do not end up modeling a specific latent in the environment are then arbitrarily assigned to an item group in the aforementioned dictionary. So if a variable's value changes independent of the candidate action performed, it is removed to ensure only relevant parts of the prior are considered.

In our experimental environment, only one surgical intervention can occur at a time, meaning the first valid intervention found is selected and executed. In environments that allow for multiple concurrent surgical interventions, different selection strategies may be employed. Since the causal model also provides uncertainty estimates for its predictions through its prior distribution [8], these can be used to choose the surgical action associated with the highest uncertainty. This approach is especially useful if the CWM is being updated with the resulting observation recieved from the environment, as it prioritizes actions that provide the most informative data about areas where the model is least certain, thus maximizing learning efficiency. This principle mirrors the strategy employed by the WAKER algorithm[21], which selects environments for agents to train on that yield the highest prediction error relative to the agent's world model, thereby allowing the agent to learn the most from that environment. Rigter et al. [21] also argue that WAKER's method qualifies as a form of intrinsic motivation, since it is goal-agnostic. Similarly, our surgical selection mechanism, being independent of any specific task or objective, can also be understood as an intrinsic motivation function.

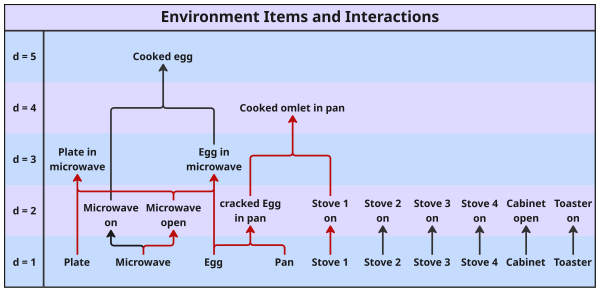

Figure 4: Visualization of the different item interactions possible in our environment. Surgical interventions can be seen in red. The higher an item appears up the interaction graph, the more an agent is rewarded for finding it, shown here as 'd'

6 Experiments and analysis

6.1 Experimental setup

6.1.1 Environment setup

Following Lippe et al. [18], we use AI2Thor as our environment, as it provides a wide range of customizable 3D rooms populated with diverse items that require specific actions to interact with. The environment naturally contains several latent causal dependencies between object, offering opportunities for the agent to discover and learn these relations. AI2Thor also includes built-in multi-agent support, making it well suited for our multi-agent setting.

We select the kitchen floor plan as our experimental environment because it contains multi-step interactions and supports the acquisition of hierarchical skills. As in Lippe et al. [18], we further restricted the environment to include only the stove setup. Although this results in a relatively small environment, it allows us to reuse the model checkpoints released by BISCUIT, with hyperparameters optimized by its authors. Despite its simplicity, this controlled setup serves as an effective testbed for evaluating BISCUIT's performance under ideal conditions, where the environment closely matches the one it was originally designed and trained for.

We introduce one novel element that BISCUIT was not trained on: a potato. This addition allows us to test how well the model handles novel objects and whether it can still predict their interactions by leveraging their causal knowledge learned from similar items. In our environment, the a potato sits on a plate, but during BISCUIT's original training it only observed how moving the plate affected both objects. It never encountered direct interactions with the potato itself. In our experiments, however, the agent is allowed to interact with the potato and the plate independently. As a result, any potato-related actions expand the domain of possible actions that BISCUIT must model and predict.

The agent interacts with the environment using two-word text commands of the form "Action Item_Name". Before sending the command to AI2Thor, the item name is parsed and replaced with the corresponding ID of an object in the frame whose name matches the item name in the command. Figure 4 illustrates all available items and the interaction pathways between them. Three of the surgical interventions possible in this environment are highlighted with red arrows. Although the environment technically supports eight surgical interventions in total, the remaining ones are either highly improbable as they require a specific setup or simply variations of the three principal cases.

Whenever an agent interacts with an object for the first time, we count it as a novel object discovery. Novelty also applies when an object changes state, for example, when an egg becomes an omelet. A novel skill is recorded when the critic marks a task as successfully completed and the corresponding action is added to skill memory. A skill is only considered novel if it is not already stored. Tracking the number of novel skills and novel item discoveries across time-steps provides a useful measure of exploration efficiency. An agent that consistently discovers more novel items and skills is learning its environment more rapidly and is therefore exploring more effectively.

Our exploration metric combines both the number of novel items n_d and the number of novel skills n_s:

$$\text{explore} = n_s + \sum_{i}^{n_d} d_i \tag{12}$$

Here, d_i is the depth of item i in the interaction graph. An item's depth is determined by the depth of the prerequisite item states needed to produce it. For example, a cooked omelet has a depth of 4, because both 'egg in pan' and 'stove 1 on' have depth of 2 and must be achieved before omelet can be created. Items with a depth 1 are those the agent can interact with directly. For reference, the sum of the depths of all possible item combinations is 208, although this theoretical upper bound is unrealistic because several state transitions are irreversible. For instance, a cooked egg cannot be returned to its raw state, meaning the agent becomes permanently locked out of some potential combinations. A high exploration score therefore indicates that the agent has uncovered multiple interactions and progressed through deeper chains of dependencies, while a low exploration score suggests limited discovery.

This design parallels the "tech ladder" used by Wang et al. [3], where proficiency in Minecraft is measured by how far an agent progresses through the game's resource hierarchy. The principle being that higher tier resources can only be collected once lower-tier ones have been acquired. In our setting, however, the focus is not on resource acquisition itself but on the causal dependencies between reusable skills that enable advancement from one rung to the next. For instance, understanding how to mine stone with a wooden pickaxe and then reusing that learned causal dependence to mine iron with a stone pickaxe reveals the agent's grasp of underlying mechanisms. To support this kind of analysis, our environment also includes items with structurally similar interactions, such as turning on a microwave and turning on a toaster.

6.1.2 Causal Model

To evaluate whether BISCUIT is a viable component for augmenting our agent with reliable causal predictions, we must first assess the accuracy of its predictions on data generated by running the Mindforge agent in the AI2Thor environment. This evaluation indicates how effectively BISCUIT can learn the environment's causal structure and use that knowledge to predict the outcomes of actions.

Our first experiment tests BISCUIT's image predictions accuracy. The model receives the previous observation X^{t-1} and the agent's action R^t, and produces a predicted future frame $\hat{X}^t$. We then compare $\hat{X}^t$ with the true future observation X^t, collected by the Mindforge agent at the next timestep. For raw image similarity, we use both Mean Squared Error (MSE) and the Structural Similarity Index (SSIM) [24]. MSE computes the average squared pixel difference, where a score of 0 indicates a perfect match and larger values indicate greater deviation. SSIM, in contrast, is designed to approximate human perceptual judgments. It compares structural information in the images rather than penalizing uniformly for pixel-level differences, making it less sensitive to global brightness shifts or other changes unlikely to be noticed by human observers.

As BISCUIT's predictions are ultimately meant to help the agent infer causal action effects, we also evaluate how well the predictions convey meaningful semantic changes. To do this, we prompt an LLM to count the number of changes between X^{t-1} and the predicted frame $\hat{X}^t$, and compare that to the number of changes between X^{t-1} and the true frame X^t. Matching change counts yield a perfect score of 1, any discrepancies reduce the score proportionally. This metric places more importance on the semantic difference related to the causal effect instead of pixel-level fidelity. Some error may arise from the LLM failing to recognize specific visual changes; lack of detection in both images is not penalized, but detecting a change in only one reduces the score.

$$\text{difference} = \frac{\text{abs}(\text{pred\_count} - \text{gt\_count})}{\text{number of predictions}} \tag{13}$$

Finally, we also evaluate BISCUIT's autoencoder reconstruction by encoding and decoding the true images and comparing the result to the original. This allows us to determine whether errors originate in the state transition model or earlier in the

disentangled representation learning stages of BISCUIT's architecture.

6.1.3 Agents

We evaluate four versions of our agent, each differing in their module composition to determine whether incorporating a causal model or a surgical action-selection function yields measurable benefits. The first agent follows the architecture proposed by Voyager [3], consisting of a curriculum module, a critic, a skill memory, and a standard action selection mechanism. This serves as our baseline for how an agent explores the AI2Thor environment without social or causal augmentation.

The second agent extends this baseline by adding the modules introduced in Mindforge [4], a belief system, episodic memory and communication capabilities. This agent provides a socially-aware baseline that incorporates reasoning about partners and a belief system structured by a Theory-of-mind causal template [5].

The third agent, Predforge, includes all Mindforge components but additionally integrates predictions from the causal model. These predictions inform the agent about the likely consequences of its contemplated actions, allowing it to factor causal effects directly into decision-making.

The fourth agent, Causalforge, also builds on the full Mindforge architecture but replaces its standard action-selection mechanism with our surgical action selection module introduced in Section 5.3. This allows the agent to deliberately intervene on specific causal components of the environment.

All four agents are evaluated in three experimental settings:

1. A single agent environment, providing a baseline for how each agent performs when the environment is influenced solely by its own actions.
2. A multi-agent environment with enabled communication, allowing for more complex interactions and more coordinated collaboration.
3. A multi-agent environment with communication disabled, testing whether agents can interpret a partner's belief's and intentions through non-verbal behavior and whether such understanding supports improved exploration.

In all multi-agent scenarios, the partner agent shares the same architecture as Mindforge. Partner agents will thus be able to communicate and maintain beliefs, but will not have access to causal model predictions or surgical action selection.

6.2 Causal model: BISCUIT

BISCUIT's image predictions are broadly accurate but tend to fail in precisely the regions that matter for causal reasoning. We evaluated BISCUIT's predictive accuracy on a set of 154 samples and found that, while its pixel-level similarity to the ground truth is generally strong, the model struggles precisely in the regions that matter most for identifying causal structure in the environment. Quantitatively, BISCUIT achieved an average MSE of 728.7 and an SSIM of 0.867, indicating that large portions of each predicted frame resemble the true observation. However, visualizing the error distribution reveals a more nuanced picture. As shown in the heatmap in Figure 5, the differences between predicted and true frames concentrate around objects and states that carry causal significance. For example, whether a stove burner is on or off, whether an egg rests on the counter, or whether a pan or held item is present. These differences, although small in number of pixels on the image, are the most problematic for our agent as it depends on these predicted states to infer causal effects.

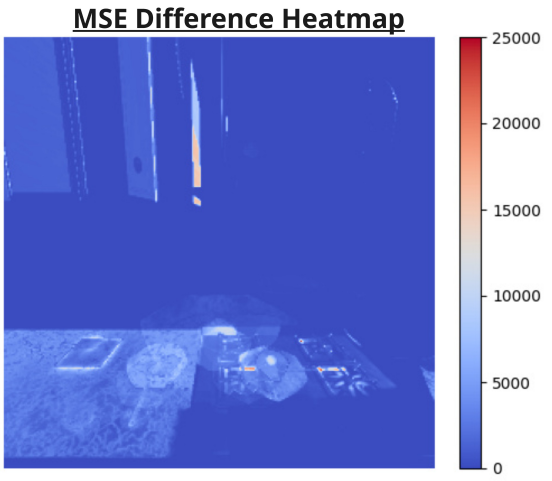

Figure 5: Visualization of where the CWM prediction errors are concentrated. The model maintains high accuracy in the stationary parts of the environment, with the exception of reflections on the counter, cabinet, and stove burner. The bright red regions correspond to these reflections, which the autoencoder fails to reproduce consistently across samples, causing them to accumulate disproportionately high MSE. In contrast, genuine prediction errors, that stem from incorrect causal reasoning, occur only once per sample. Their pixel-wise contributions are therefore averaged out much more, appearing as the light blue outlines around semantically relevant areas of the scene. These include regions indicating whether the bottom-right stove is on or off, whether the pan and egg are located on the stove or counter, and the central zone where held items typically appear.

Some of these high-error regions arise from limitations in the autoencoder rather than from errors in causal reasoning. Reflections on metal surfaces, such as the stove or the countertop, are rarely reconstructed accurately, leading BISCUIT to produce

dimmer surfaces that are heavily penalized by MSE despite being semantically irrelevant. Nonetheless, the heatmap also reveals more meaningful differences. Faint "ghost" outlines of a pan or egg on the counter show that BISCUIT sometimes fails to predict interactions involving these items, and similar errors appear when the model attempts to predict the presence of an omelet in a pan or the activation state of the bottom-right stove burner. Errors also cluster around the center of the screen where held items appear, indicating persistent difficulty in modeling held-object interactions.

BISCUIT's autoencoder faithfully reconstructs static scenery but often fails on the causal state of some of the key objects in the image. To evaluate how much of BISCUIT's prediction error originates from its autoencoder rather than its transition model, we compared each image to its own encoded–decoded reconstruction. Across 154 samples, this yielded an MSE of 681.4 and an SSIM of 0.871, values remarkably close to those obtained for full prediction accuracy. These results indicate that a substantial portion of the overall prediction error is introduced before any action-conditioned reasoning even occurs. Visualizing reconstruction differences reveals a heatmap almost identical to that seen in Figure 5, with errors clustering around precisely the areas that encode the environment's causal state.

This pattern reflects how the autoencoder was trained. It was exposed only to images from this single room and a fixed camera perspective, allowing it to reproduce stationary background surfaces such as walls, cabinets, countertops—with near-perfect consistency. However, it remains unreliable at reconstructing the dynamic, semantically meaningful parts of the environment. Object states that are part of causal links, such as whether a stove burner is on or off or whether an egg is present on the counter, can end up being flipped. Stoves in particular are often toggled incorrectly in the reconstruction. The relevant prediction errors appear light blue instead of red in Figure 5, as each type of false prediction does not appear every sample like the reflection errors do and thus get averaged out a lot more.

LLM-based evaluation reveals that BISCUIT's semantically meaningful prediction errors are caught by the agent. To measure how well BISCUIT's predictions preserve the causal effects of an action and transmit it to the agent, we asked an LLM to identify the number of meaningful changes between the previous frame and both the predicted frame and the true next observation. Across 154 prediction–original observation pairs, the changes the LLM detected differed on average by 1.74 from the number of changes detected on the ground-truth-original observation pairs, indicating that BISCUIT frequently introduces or removes state changes from the environment.

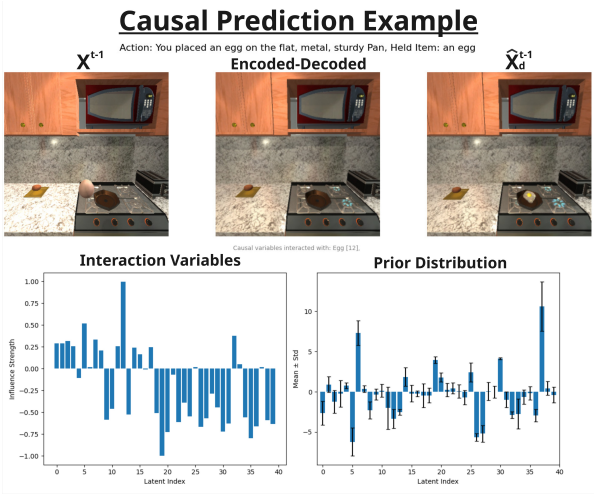

Figure 6: Example of a prediction by the Causal World Model. The top three images show the original environment observation, the observation encoded then decoded again, and finally the prediction. The envisioned action here is to place the egg into the pan. The interaction variables are shown as bar chart on the bottom left, and the prior distribution from which the prediction is sampled is shown on the bottom right.

Although the LLM often failed to detect subtle changes, such as a stove toggling on or off, it reliably flagged larger, more visually prominent differences. For instance, predictions commonly omitted held items which was almost always captured by the LLM. Crucially, the same LLM is used later by our Predforge and Causalforge agents to reason about the effects of their actions in the prediction. Any discrepancy flagged here therefore translates directly into misinformation for those agents in subsequent experiments, negatively impacting their decision making.

BISCUIT's action grammar compensates for autoencoder failures, preserving crucial information about held items. Although the autoencoder struggles to reconstruct objects currently being held, often causing them to vanish in the encoded–decoded frame, these errors are often corrected once the action is processed through BISCUIT's specialized action grammar. Before reaching the transition model, every textual action is rewritten into a richer, semantically grounded form that explicitly encodes the held object. For instance, an action like "PutObject Pan" is transformed into "You placed an egg on the flat, metal, sturdy Pan," ensuring that the transition model still receives clear information about what is being manipulated, even if the autoencoder fails to render the held item correctly.

This effect is illustrated in Figure 6: the egg disappears in the autoencoder's reconstruction but reappears in the prediction as a cracked egg in the

Figure 7: Visualization of running Voyager, Mindforge, Predforge and Causalforge agents in the single agent environment. Expectation score (green) averaged over 20 runs is shown per timestep where a higher exploration score means the agent discovered more causal interactions with the environment as per Equation 12. Novel items (orange) and skills (blue) found are also shown.

pan, indicating that the transition model inferred the correct outcome despite the missing visual cue. The interaction variables shown in the bottom-left of the figure inform us that the action interacted with variable 12, as it crosses the experimentally determined interaction threshold of 0.75, separating genuine object interactions from latent noise. According to BISCUIT's dictionary, variable 12 does in fact correspond to the egg. Given the change in the environment, we can assume it is responsible for determining whether the egg is intact or broken in the pan. This interaction then changes the prior distribution to what is seen on the bottom-right of Figure 6 from which the prediction is sampled.

6.3 Single Agent setting

All agents achieve comparable exploration efficiency in the single-agent environment, with Voyager and Predforge performing slightly better and Causalforge slightly worse. Across the single-agent setting, all four agents exhibit a broadly similar exploration trajectory, uncovering new items and skills in a smooth, steadily increasing manner. Voyager and Predforge reach the highest exploration scores around 22, as per Equation 12. While Mindforge finishes slightly lower, just under 19, and Causalforge settles at roughly 17.5. Mindforge's reduced performance appears linked to its additional architectural complexity. In a small environment with only a handful of valid interactions, the larger number of LLM calls increases the risk of hallucinated beliefs and actions toward items that seem logical but do not actually exist in this environment. Its reasoning systems are designed for richer, more varied multi-agent worlds, and thus become counterproductive when forced into a highly constrained setting. Predforge, however, mitigates some of these issues by integrating causal predictions, which help guide its decisions and prevent certain hallucination-driven mistakes, even though it retains the same underlying architecture as Mindforge.

Despite these differences, all agents display a logarithmic exploration curve, rapidly discovering the most obvious interactions early on and then slowing as the remaining unexplored interactions become harder to identify. This pattern suggests that each agent quickly exhausts the shallow parts of the environment's "tech tree" before hitting the diminishing-returns phase of exploration that require more complex interactions to attain.

Surgical interventions reduce CausalForge's efficiency in acquiring and saving new skills. Although CausalForge discovers the same number of novel items as Mindforge, it records noticeably fewer novel skills, resulting in a lower overall exploration score. This drop stems from the behavior of the surgical intervention module, which selects and performs interventions independently of the arbitrary task the agent is currently pursuing. Performing an intervention directly comes with an initial penalty to the immediate exploration efficiency. By adding an intervention action, the current interaction takes an additional iteration to complete. As interventions are not selected with the current task in mind, completing the intervention will not necessarily lead to the tasks success. In that scenario, any skills learned from the intervention will not be stored in skill memory or count towards the skill metric.

A second contributing factor is the metric's reliance on the critic's judgment of task success. Skill acquisition only occurs when the critic declares that a task has been completed, meaning any hallucinated successes, or missed detections of genuine success, directly affect whether a skill is saved and counted in the metric.

Novel item discoveries do not reliably translate into acquired skills. Across agents, the number of novel item interactions consistently exceeds the number of skills learned, even after accounting for item depth. This gap arises because the agent often assigns itself tasks that are impossible in the environment. While pursuing these unattainable goals, it still performs various exploratory "random" interactions that lead to novel item discoveries, but these interactions do not contribute to task completion and therefore are never saved as skills. As a result, valuable environment knowledge may be uncovered without being retained.

CausalForge identifies only two types of surgical interventions. Across intervention opportunities, CausalForge successfully detected a valid surgical intervention in 59% of cases, but only for

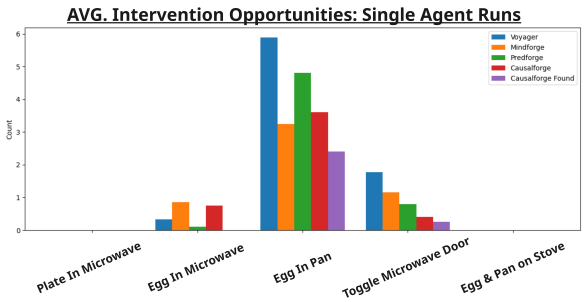

Figure 8: Visualization of average number of surgical intervention opportunities encountered by each agent and for each type of surgical intervention in the single agent environment. A high number count means that type of intervention is often encountered in the environment, a low count means agents rarely had the opportunity to perform this intervention. Voyager is shown in blue, Mindforge in orange, Predforge in green, Causalforge in red and the amount of times Causalforge successfully found a surgical intervention when one was available in Purple.

Figure 9: Visualization of running Voyager, Mindforge, Predforge and Causalforge agents the in Multi-agent environment with one other partner agent. The figure displays the score of only the main agent. Expectation score (green) is shown per timestep where a higher expectation score means the agent discovered more causal interactions with the environment as per Equation 12. Novel items (orange) and skills (blue) found is also shown.

two categories: "Egg in Pan" and "Toggle Microwave Door". As shown in Figure 8, the overwhelming majority of opportunities encountered by every agent was "Egg in Pan". This bias stems from the agents' strong tendency to interact with the egg, which most agents select as their initial task, and picking up the egg is often the very first action taken. This behavior can be explained by the examples used in the prompts for modules such as the critic and auto-curriculum. As in Voyger[3] and Mindforge [4], module prompts contain environment examples to give the LLM a better idea of what output is expected. Our examples often refer to an egg, which implicitly encourages egg-centered interactions in modules such as the auto-curriculum and critic. The remaining intervention types such as "Plate in Microwave" and "Egg and Pan on Stove" were never discovered by any agent.

The limited environment offers too few meaningful intervention pathways for surgical actions to improve exploration. The small number and low diversity of surgical interventions available in this restricted environment prevented CausalForge from converting its successful detections into measurable exploration gains. As illustrated in Figure 8, the handful of interventions identified involving the egg or the microwave door, rarely contribute useful causal insights for future interactions. In such a compact setting, understanding a dependency (e.g., how opening the microwave door affects its state) offers little leverage for discovering deeper or more complex item relationships.

Moreover, performing a surgical intervention typically costs a timestep, making it less efficient than simply executing the task directly. For example, cooking the egg immediately rather than intervening on the microwave beforehand results in the novel item "cooked egg" one iteration earlier. The environment's causal structure contains few multi-step dependencies that would allow surgical knowledge to compound further up the "tech ladder," limiting the practical value of intervention-based reasoning. A larger, more interconnected environment, or updating the CWM with new knowledge gained from interventions, would likely be necessary for surgical actions to yield measurable exploration benefits.

6.4 Multi-agent setting with communication

Agents achieve higher exploration scores in the multi-agent setting with communication, but relative performance remains similar. Figure 9 shows the performance of all four agents in a multi-agent environment paired with a single partner agent, with communication enabled. In this setup, all agents reached exploration scores of approximately 30, representing a noticeable improvement over their single-agent performance. The boost is largely due to the ability coordinate actions, which helps agents avoid redundant exploration and speeds up the discovery of novel items and skill by combining efforts. Communication also helped mitigate weaknesses observed in the single-agent setting for Mindforge and CausalForge.

The increase in exploration score could also stem from a failure case being fixed, where agents repeatedly targeted the wrong counter when attempting to place down the plate. Each counter in the environment has a unique alphanumeric ID, and without guidance, guessing the correct counter was essentially impossible. In the multi-agent experiments, the environment interface was adapted so that the intended counter was always targeted,

Figure 10: Visualization of running Voyager, Mindforge, Predforge and Causalforge agents the in Multi-agent environment with one other partner agent. The figure displays the score of both agents summed together. Expectation score (green) is shown per timestep where a higher expectation score means the agent discovered more causal interactions with the environment as per Equation 12. Novel items (orange) and skills (blue) found is also shown.

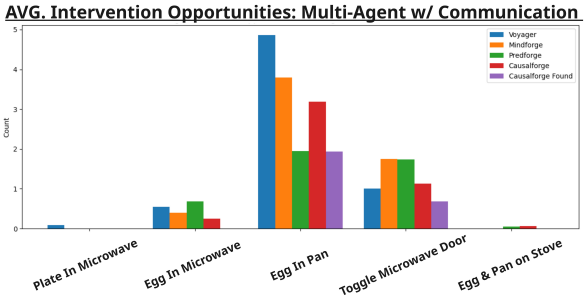

Figure 11: Visualization of average number of surgical intervention opportunities encountered by each agent and for each type of surgical intervention in the multi-agent environment with communication enabled. A high number count means that type of intervention is often encountered in the environment, a low count means agents rarely had the opportunity to perform this intervention. Voyager is shown in blue, Mindforge in orange, Predforge in green, Causalforge in red and the amount of times Causalforge successfully found a surgical intervention when one was available in Purple.

eliminating this specific source of failure. However, this single fix alone cannot account for the overall increase in exploration scores, nor does it alter the relative boost in performance received by Mindforge and Causalforge agents when compared to the single agent experiment.

Considering both agents' discoveries substantially boosts exploration scores. Figure 10 shows the combined exploration of the main agent and its partner for all four agent types in a multi-agent environment with communication enabled. When summing the novel items found and skills acquired by both agents, the overall exploration score jumps to around 43, highlighting that the partner agent uncovered interactions that the main agent alone did not encounter. The faster initial rise of the curves further indicates that the partner agent often discovered items earlier, contributing to higher exploration scores at earlier time-steps and demonstrating the benefits of collaborative exploration.

Voyager struggles to acquire skills due to lack of partner modeling. Voyager shows only a minimal increase in skills learned compared to its single-agent performance. Its inability to model the partner's intentions or reasoning prevents it from effectively learning from the partner agent's discoveries. This contrasts with the other agents, which show slight gains in skill acquisition when operating collaboratively. The discrepancy is further confirmed by the joint exploration metrics in Figure 10. The partner Mindforge agent contributes a significant number of novel skills, which Voyager alone fails to capture, emphasizing the importance of partner modeling for skill learning in multi-agent settings.

CausalForge's ability to identify surgical interventions remains unchanged between single and multi-agent environments. Figure 11 illustrates the number of surgical intervention opportunities of each type encountered by all agents in the multi-agent environment. CausalForge did encounter the "Egg and Pan on Stove" intervention during the multi-agent run. However, the frequency was too low to confidently attribute this to the multi-agent setting. It is likely that, given more trials, this intervention would also appear in the single-agent environment. Importantly, CausalForge was ultimately unable to successfully identify and execute this intervention, demonstrating that the multi-agent context did not materially improve its ability to find or perform surgical interventions.

6.5 Multi-agent setting no communication

Removing communication boosts Voyager's performance. Figure 12 shows the performance of all four agents in a multi-agent environment with one partner agent, but with communication disabled. In this setting, Voyager achieves an exploration score of around 37, outperforming the other agents, which score of approximately 30.

We do not observe much of a drop in the Mindforge based agents when communication is removed. This could be because the available task in the environment are too small to reward collaboration, most of them can directly be accomplished by one agent. This diminishes the benefit one would expect from communication at being able to collaborate and share beliefs. Voyager does not maintain

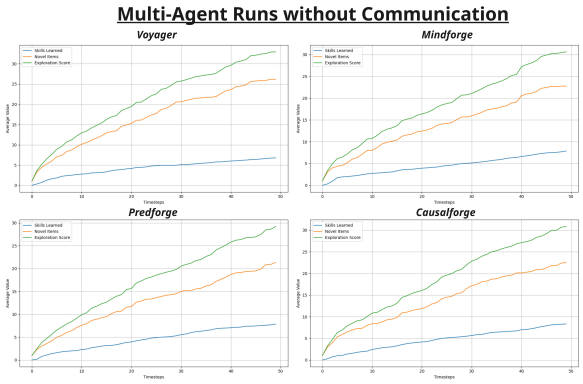

Figure 12: Visualization of running Voyager, Mindforge, Predforge and Causalforge agents the in the multi-agent environment without communication. The figure displays the score of only the main agent. Expectation score (green) averaged over 20 runs is shown per timestep where a higher expectation score means the agent discovered more causal interactions with the environment as per Equation 12. Novel items (orange) and skills (blue) found is also shown.

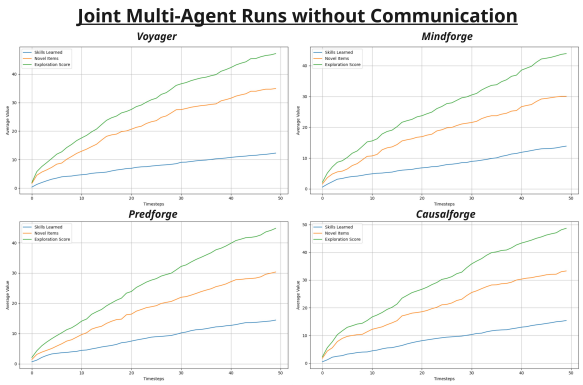

Figure 13: Visualization of running Voyager, Mindforge, Predforge and Causalforge agents in the multi-agent environment without communication. The figure displays the score both agents summed together. Expectation score (green) is shown per timestep where a higher expectation score means the agent discovered more causal interactions with the environment as per Equation 12. Novel items (orange) and skills (blue) found is also shown.

partner-dependent beliefs, so removing communication allows it to focus solely on its own tasks. Additionally, disabling communication eliminates the risk of "hallucinated" information being shared between agents. For example, if a partner falsely believes that bread exists in the environment and communicates their attempts to pick it up, this could mislead Voyager into wasting time searching for a non-existent object. Without communication, Voyager avoids such distractions and can explore more efficiently.

Without communication, agents cannot share discoveries, creating disparities in joint exploration scores. Figure 13 shows the combined exploration scores of the main agent and its partner when communication is disabled. Voyager maintains a high joint exploration score of around 48, while Mindforge remains slightly lower at approximately 40. Interestingly, Predforge sees the largest increase in joint exploration, reaching around 46, suggesting that the partner agent contributed substantially more than the main agent. This disparity arises because, without communication, agents are unable to share discoveries, leading to uneven exploration contributions between the interacting agents.

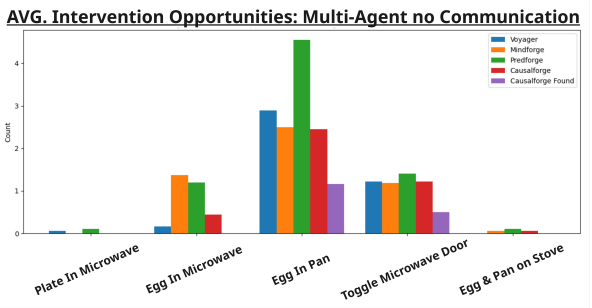

Figure 14: Visualization of average number of surgical intervention opportunities encountered by each agent and for each type of surgical intervention in the multi-agent environment without communication. A high number count means that type of intervention is often encountered in the environment, a low count means agents rarely had the opportunity to perform this intervention. Voyager is shown in blue, Mindforge in orange, Predforge in green, Causalforge in red and the amount of times Causalforge successfully found a surgical intervention when one was available in Purple.

Multi-agent settings may encourage agents to explore previously unvisited areas of the environment. Figure 14 shows the surgical intervention opportunities encountered by agents in the multi-agent environment when communication is disabled. As in the other experimental settings (Figures 8 and 11), "Egg in Pan" is the most frequently encountered intervention, followed by "Toggle Microwave Door" and "Egg in Microwave". Importantly, all types of interventions were found by the agents in this setting. This suggests that the presence of a second agent can push the main agent to explore areas of the environment that its partner is not interacting with, enabling the discovery of novel environment states and surgical interventions that were not encountered in the single-agent experiments. This effect is particularly likely in small environments, where agents naturally gravitate toward the same objects, making partner-driven exploration a key factor in inducing new interactions.

6.6 Failure cases

CausalForge can only detect two types of interventions, BISCUIT's limitations prevent others from being identified. To better understand CausalForge's ability to discern surgical interventions, we manually constructed each intervention scenario and evaluated whether the combined BISCUIT + surgical-intervention module could correctly classify them as valid interventions. Across the different cases, we find that failures largely stem from BISCUIT's inability to accurately detect interaction variables or predict key causal state changes, as well as from limitations in how the intervention-selection function detects changes in the prior.

For the intervention "put the plate in the microwave," no causal variable in the prior appeared to change as a result of the action. Because the intervention-selection mechanism depends on detecting changes in individual causal variables, it concluded that the intervention had no effect and rejected it. This could indicate that BISCUIT failed to predict the effect, or that the prior-change detection method, which is checking causal variables independently, is too simplistic. While no single microwave or plate related variable changed significantly, it is possible that the change is encoded across combinations of variables. Updating the detection function to compare joint variable configurations, rather than checking variables independently, may allow more subtle prior shifts to be recognized.

For "put the egg in the microwave," BISCUIT failed to identify any interaction variable associated with the egg or the microwave. Examination of the interaction-variable graph shows no prominent spike indicating an interaction. Because the surgical intervention module relies on identifying a clear interaction variable before checking the prior, it was unable to classify this scenario as a valid surgical intervention.

For "place the pan containing the egg on the stove," BISCUIT correctly identifies the interaction with the pan but does not detect that the action also involves the egg. As a result, the egg's causal variables are never inspected for prior changes. Even though toggling the stove on would immediately cook the egg, a valid and meaningful causal intervention, the missing egg interaction prevents CausalForge from checking the prior for egg changes, thus not recognizing the intervention as valid.

In the case of "close the microwave door before toggling it," BISCUIT's transition model fails to represent the dependency between the microwave door and the microwave's ability to activate. Regardless of whether the door was originally open, the transition model simply predicts the microwave as closed and turned on. This prevents the agent from recognizing any dependencies between the door state and any changes in the final prior, thus categorizing

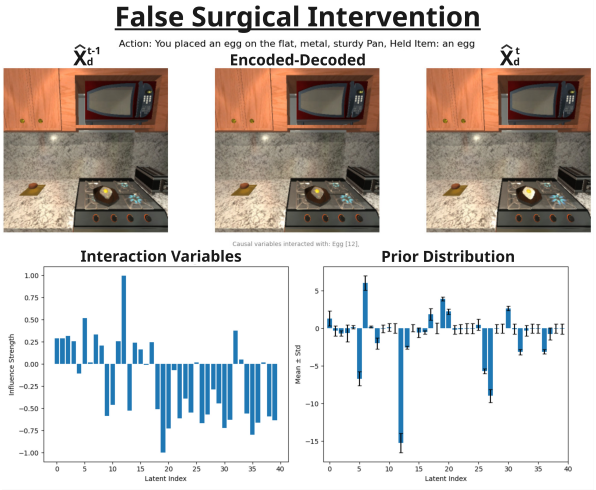

Figure 15: Presented is an example of a invalid surgical intervention that got classified as valid due to incorrect BISCUIT predictions. Shown is the second part of the two step prediction process for identifying surgical interventions explained by Equation 11. The first step referring to Equation 10 can be seen in Figure 6. The first image is the prediction made by BISCUIT in the first prediciton step. The second image is that same image encoded then decoded by BISCUIT's autoencoder, the last image is the final two step prediction used to descern the effect of a full surgical intervention

the intervention as invalid.

CausalForge does successfully identify the intervention "OpenObject Microwave" before attempting to open it. In this case, the action flips the door to its opposite state, a pattern the agent correctly learns. While this knowledge is not particularly useful for discovering new items or acquiring novel skills, it demonstrates that CausalForge can learn certain reversible relationships when BISCUIT provides a reliable signal.

Finally, CausalForge also correctly detects the intervention "place the egg in the pan," where toggling the stove before placing the egg causes it to cook instantly. Here, BISCUIT successfully identifies the correct interaction variable for the egg and updates the prior to reflect the causal effect. This allows the intervention-selection module to classify it as a valid surgical intervention. This is one of the only cases where the entire causal chain, from interaction detection to prior comparison, functions as intended.

False positives arise in the "egg in pan" scenario due to BISCUIT predicting incorrect causal effects. False positives also appear in the "egg in pan" intervention case, as illustrated in Figure 15. This figure shows the second step of the surgical-intervention from Equation 11 prediction pipeline, while the first step, referring to Equation 10, can be seen in Figure 6. The false positive occurs when both the main action and the proposed

intervention attempt to place the egg into the pan. Even though the stove beneath the pan is off, BISCUIT nevertheless predicts that the egg becomes cooked and updates the prior accordingly. As can be seen in the interaction-variable graphs at the bottom-left of both figures, BISCUIT's interaction variables depend deterministically on the text action itself and does not change depending on the current state of the environment. This means that the same action always activates the same set of interaction variables. This is not only true for this case, but all possible actions and their interaction variables. Here, both the main action and the intervention activate variable 12, associated with the egg, regardless of whether the egg is already in the pan or whether the stove is on. Coupling this with the fact that the prior does change in BISCUIT's prediction, the system incorrectly labels the intervention as a valid surgical intervention.

This behavior likely stems from BISCUIT's training data lacking examples where the agent attempts to place the egg in the pan when it is already there. Notably, this false positive and the genuine "egg in pan" intervention are usually discovered together, but the false positive often appears first, meaning it is the one CausalForge typically executes in the environment. Implementing a more sophisticated decision rule for selecting among multiple candidate interventions, rather than choosing the first one marked valid, would help mitigate this issue and improve the reliability of the intervention-selection process.

Additional failure cases stem from perception errors and restrictive environment dynamics. Beyond issues specific to surgical interventions, several broader failure modes arise from limitations in both the agent's perception system and the AI2-THOR environment itself. The vision language model (VLM) occasionally misidentifies objects. For example, frequently labeling the potato as bread. These misperceptions propagate through the agent's modules making the autocurriculum proposes tasks involving "bread", which guides the agent to attempt action commands involving bread. This leads to the agent repeatedly attempting interactions with an object that does not exist. Since such actions never succeed, they waste time, introduce noise into the agent's beliefs, and reduce exploration efficiency.

The environment also imposes strict and sometimes unintuitive constraints that contribute to failure. Certain actions that appear logically feasible to a human, such as placing the egg inside an open cabinet, are simply not allowed in AI2-THOR's environment. As a result, agents often pursue tasks that seem reasonable within their llm-inferred world model but are impossible to execute in practice. These cases also waste time which reflects as a lower exploration score.

7 Conclusion

This work introduced two new causally-aware agent architectures, Predforge and CausalForge, that extend Voyager and Mindforge with the ability to anticipate the effects of actions through BISCUIT's causal world model and, in the case of CausalForge, to actively search for and perform surgical interventions during exploration. To our knowledge, this is the first attempt to introduce surgical causal interventions within an LLM-driven embodied agent.

Our experiments show that BISCUIT can produce visually plausible predictions, but its errors often occur in the very regions that carry causal significance, limiting the reliability of intervention detection. Nevertheless, CausalForge successfully identifies specific classes of interventions, demonstrating that the architecture can function end-to-end when BISCUIT provides the necessary causal signals. Across different settings, we find that partner agents boosts overall exploration for all agents, but removing communication actually improves performance in Voyager agents which are not equipped with a specific collaboration system.

By analyzing both successful and failed interventions in detail, we highlight several key limitations in the causal model, the environment, and the agent pipeline. These findings outline clear opportunities for future work, including online updating of the causal model, richer environments with more reusable interventions, improved partner modeling in the causal world model, and refined skill learning mechanisms. Together, these results establish a foundation for more powerful causal exploration methods in LLM-driven embodied agents.

7.1 Future work

Several directions could strengthen the effectiveness of causal reasoning and intervention-driven exploration. First, the knowledge gained from performing surgical interventions is not yet used to improve BISCUIT's predictions. Enabling online updates, where post-interaction environment observations serve as ground-truth corrections, could allow the CWM to continually refine its causal links and adapt to novel environment dynamics. Second, expanding the environment with a richer set of interventions, items, and multi-step tasks would create opportunities for causal knowledge to become relvant, turning interventions from short-term penalties into long-term accelerators of exploration. Having compounding tasks within the environment that give agents a better opportunity to collaborate would also be interesting to see for the multi-agent environments. Third, Mindforge's partner-belief representations could be passed into the CWM alongside environmental embeddings, enabling more accurate predictions in multi-agent scenarios where another agent's behavior plays a causal role. Fourth, an informative ablation study could

bypass CWM prediction errors by directly feeding detected intervention opportunities into the auto-curriculum, allowing us to test whether surgical interventions truly boost exploration when identified correctly. Integrating the intervention directly into the task generation system ensures that any discoveries arising from a surgical intervention are saved as skills. Another possibility is to decouple skill saving from the task entirely, allowing the agent to store a skill whenever it performs a meaningful interaction, even if it falls outside the current task. This would provide a more faithful measure of competence acquisition and could accelerate overall exploration.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[3] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL `https://arxiv.org/abs/2305.16291`.

[4] Mircea Lică, Ojas Shirekar, Baptiste Colle, and Chirag Raman. Mindforge: Empowering embodied agents with theory of mind for lifelong collaborative learning, 2024. URL `https://arxiv.org/abs/2411.12977`.

[5] Kanishk Gandhi, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D. Goodman. Understanding social reasoning in language models with language models, 2023. URL `https://arxiv.org/abs/2306.15448`.

[6] Aneesh Komanduri, Xintao Wu, Yongkai Wu, and Feng Chen. From identifiable causal representations to controllable counterfactual generation: A survey on causal generative modeling, 2024. URL `https://arxiv.org/abs/2310.11011`.

[7] Yoshua Bengio. Reusable modular and causal knowledge representation for lifelong learning. In *CoLLAs*, 2022.

[8] Phillip Lippe, Sara Magliacane, Sindy Löwe, Yuki M Asano, Taco Cohen, and Efstratios Gavves. Language agents meet causality – bridging LLMs and causal world models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=y9A2TpaGsE`.

[9] Taco Cohen. Towards a grounded theory of causation for embodied ai, 2022. URL `https://arxiv.org/abs/2206.13973`.

[10] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[11] Nicholas K Humphrey. The social function of intellect. In *Growing points in ethology*, pages 303–317. Cambridge University Press, 1976.

[12] John E Laird. *The Soar cognitive architecture*. MIT press, 2019.

[13] Anand Rao and Michael Georgeff. Bdi agents: From theory to practice. 11 2000.

[14] Tobias Gerstenberg and Joshua B Tenenbaum. Intuitive theories. 2017.

[15] PATRICK SUPPES, HM Blalock, A Aganbegian, FM Borodkin, R Boudon, and V Capecchi. A probabilistic analysis of causality. *Quantitative Sociology: International Perspectives on Mathematical and Statistical Modeling*, page 49, 1975.

[16] David Lewis. Causation. *The journal of philosophy*, 70(17):556–567, 1973.

[17] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

[18] Phillip Lippe, Sara Magliacane, Sindy Löwe, Yuki M Asano, Taco Cohen, and Efstratios Gavves. Biscuit: Causal representation learning from binary interactions. In *Uncertainty in Artificial Intelligence*, pages 1263–1273. PMLR, 2023.

[19] Sébastien Lachapelle and Simon Lacoste-Julien. Partial disentanglement via mechanism sparsity. *arXiv preprint arXiv:2207.07732*, 2022.

[20] Olivier L. Georgeon, James B. Marshall, and Simon Gay. Interactional motivation in artificial systems: Between extrinsic and intrinsic motivation. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–2, 2012. doi: 10.1109/DevLrn.2012.6400833.

[21] Marc Rigter, Minqi Jiang, and Ingmar Posner. Reward-free curricula for training robust world models, 2024. URL `https://arxiv.org/abs/2306.09205`.

[22] Chao Yu, Minjie Zhang, and Fenghui Ren. Emotional multiagent reinforcement learning in social dilemmas. In *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, pages 372–387. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-44927-7.

[23] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A. Ortega, DJ Strouse, Joel Z. Leibo, and Nando de Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning, 2019. URL `https://arxiv.org/abs/1810.08647`.

[24] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE*

Transactions on Image Processing, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

[25] Google Google team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL `https://arxiv.org/abs/2507.06261`.

[26] Phillip Lippe, Sara Magliacane, Sindy Löwe, Yuki M Asano, Taco Cohen, and Stratis Gavves. Citris: Causal identifiability from temporal intervened sequences. In *International Conference on Machine Learning*, pages 13557–13603. PMLR, 2022.

[27] Phillip Lippe, Sara Magliacane, Sindy Löwe, Yuki M Asano, Taco Cohen, and Efstratios Gavves. Causal representation learning for instantaneous and temporal effects in interactive systems. *arXiv preprint arXiv:2206.06169*, 2022.

A Agent Architecture

Listed are the different main modules used in our agents. We also provide the prompts used in each modules with their examples (in {{yellow}}) and variables containing information was integrated into them (in {red}). The LLM used for all our experiments was Gemini 2.0 flash by Google[25].

A.1 Voyager Modules

The 'Voyager' agent used in this paper follows the same architecture as the one put forwards by the original Voyager[3] paper. An iteration unfolds in quite the same way as Figure 3, except only the auto-curriculum, critic, skill memory and action selection modules A.3 are used.

A.1.1 Auto-Curriculum

The goal of the auto-curriculum is to generate simple tasks that the agent could be able to achieve in the environment given its current resources and acquired skills. The auto-curriculum takes as input: the current image frame, the last action performed by the agent, the last thoughts of the agent, whether the last task was succeeded or failed, the critique generated by the critic if the task was indeed failed, any communications with other agents and the item currently being held. It then returns a task for the agent to perform in text format. It also internally keeps track of what tasks have been succeeded or failed, so as not to needlessly repeat succeeded tasks or keep trying tasks that always end in failure.

The auto-curriculum starts by generating a list of questions linked to the agent and its environment using the following prompt and the current image frame:

Auto_Curriculum, questions generation

You are a helpful assistant that asks questions to help me decide the next immediate task to do. My ultimate goal is to discover as many diverse things as possible, accomplish as many diverse tasks as possible and become the best robot in the world.
Completed tasks so far: **{completed_tasks}**
Failed tasks that are too hard: **{failed_tasks}**
Last communications received from other agents: **{communications}**
I will also provide you with an image of what the robot is currently looking at.
You must follow the following criteria:
1) You should ask at least 5 questions (but no more than 10 questions) to help me decide the next immediate task to do. Each question should be followed by the concept that the question is about.
2) Your question should be specific to a concept relevant to the current environment.
Bad example (the question is too general): {{*Question: What is the best action to do in the environment?, Concept: unknown*}}
Bad example (cooking is still general, you should specify the type of cooking such as cooking egg): {{*Could the egg be cooked in the pot?, Concept: cooking*}}
Good example: {{*Question: How would you slice bread?, Concept: sliced bread*}}
3) Your questions should be self-contained and not require any context.
Bad example (the question requires the context of my current room):{{*Question: What are the items that I can find in my current room?, Concept: unknown*}}
Bad example (the question requires the context of the environment):{{*Question: Do you have knife next to you?, Concept: knife*}}
Bad example (the question requires the context of my nearby items):{{*Question: Is there a stove nearby?, Concept: stove*}}
Good example:{{*Question: What would be the items that I could find in the fridge?, Concept: fridge contents*}}
4) Do not ask questions about other rooms or leaving the room (such as opening or seeing what is behind the door) since they are too hard for me to do and I must stay in the current room.
Let's say you see a fridge in the image. You can ask questions like: {{*Question: How could I iteract with the fridge?, Concept: fridge*}}
{{*Question: What could I find inside the fridge?, Concept: fridge contents*}}
Let's say other agents have communicated to you that they are trying to cook an egg. You can ask a question like: {{*Question: What could I do to help the other agent cook an egg?, Concept: help cook egg*}}
Let's say your last completed task is "pick up a knife". You can ask a question like: {{*Question:*

What are the suggested tasks that I can do after having picked up a knife?, Concept: knife}}
Here are some more question and concept examples: {{ *Question 1: What can I do witha slice of bread?, Concept 1: slice of bread, Question 2: How could I turn on the stove?, Concept 2: stove, Question 3: How could I use the toaster to create new items and learn new skills?, Concept 3: toaster, Question 4: How to obtain an egg?, Concept 4: egg, Question 5: What are the benefits of using a pot on top of the stove?, Concept 5: stove pot, Question 6: What new items require a potato to be made?, Concept 6: potato,* }}
IMPORTANT, You should only respond in the format as described below, where field 'questions' is a list of strings:
RESPONSE FORMAT: {{ *reasoning: ..., questions: ["Question 1: ...", "Concept 1: ...", "Question 2: ...", "Concept 2: ...", "Question 3: ...", "Concept 3: ...", ..., "Question n: ...", "Concept n: ...",]* }}

The list of questions is then answered one at a time by the LLM using the following prompt and the image frame:

Auto_Curriculum, answering question

You are a helpful assistant that answer my question about the environment.
You will answer a question and based on the context (only if available and helpful), an image frame of whats around you and your own knowledge of the environment. Always give the simplest answers when multiple answers are possible.
1) IMPORTANT always answer in the following JSON format: {{*"answer": answer*}}
2) Answer "answer: Unknown" if you don't know the answer.
Question: **{question}**
Relevant Past Contexts: **{relevant_past_context}**

The goal of this process is to widen the perspective of the LLM, in order for it to truly take in account everything possible in the environment before it generates the new current task. This reduces the chance that an agents keeps focusing on the same part of the environment, even if it is not working. Answering the questions also helps the LLM gauge how 'simple' of a task it would be for the agent, increasing the probability the agent could actually complete it.

The question answer pairs, as well as the rest of the auto-curriculum input, is then passed with the following prompt to the LLM:

Auto_Curriculum, task selection

You are a helpful assistant that tells me the next immediate task to do. My ultimate goal is to discover as many diverse things as possible, accomplish as many diverse tasks as possible and become the best robot in the world.
I will give you the following information:
{question_answers}
Completed tasks so far: **{completed_tasks}**
Failed tasks that are too hard: **{failed_tasks}**
Last task: **{last_task}**
Last action: **{last_action}**
Previous thoughts: **{last_thoughts}**
Last task was a success: **{success}**
Critique: **{critique}**
Held item: **{picked_object}**
Last communications received from other agents: **{communications}**
I will also provide you with an image of what the robot is currently looking at.
You must follow the following criteria:
1) You should act as a mentor and guide me to the next task based on my current learning progress.
2) Please be very specific about what items I need to collect, what I interaction I need to do with them.
3) The task should involve doing some action(s) inside the environment. Do not ask me to

"describe" or "think".
4) The next task should follow a concise format, such as {{*"Pick up [item]"*}}, {{*"Put [item] in [item]"*}}, {{*"Slice [item]"*}}, {{*"Bring [item] to [item]"*}}, {{*"Open [item]"*}}, {{*"Move to [area]"*}} etc. It should be a single phrase. Do not propose multiple tasks at the same time. Do not mention anything else.
5) The next task should not be too hard since I may not have the necessary resources or have learned enough skills to complete it yet.
6) The next task should be novel and interesting. I should look for new items, find unique interactions with items, and discover new things. I should not be doing the same thing over and over again.
7) I may sometimes need to repeat some tasks if I need to collect more resources to complete more difficult tasks. Only repeat tasks if necessary.
8) If the last task was a failure, the new task MUST BE DIFFERENT from the old task.
You should only respond in the format as described below: IMPORTANT always answer in the following JSON format:
{{ *"reasoning": Based on the information I listed above, do reasoning about what the next task should be, "task": The next task* }}
Here's an example response: {{ *"reasoning": "I see a piece of sliced bread and a toaster", "task": "Put bread in toaster"* }}

The task generated by the LLM is then set as the current task and returned to the agent. The generated tasks do have a chance to contain hallucinations or to be too complex for the agent to be able to succeed, but these series of prompts templated by Voyager [3] help in reducing such cases.

A.1.2 Critic

The critic analyses the returned environment observation to determine if the current task was achieved. It takes as input: the current image frame, the last action performed, the current task, any communication with other agents, currently held items and any returned errors from the environment. Note that for Mindforge based agents, the critic also receives the current task beliefs through variable 'context'. In Voyager however, the value of that variable will be 'None'. The critic outputs a boolean value signifying whether the task was successfully achieved. If it was not achieved, it also returns a 'critique' proposing different tips as to what might of gone wrong and how to succeed with the next action. This is generated by sending the following prompt to the LLM:

Critic, success determination and "critique" generation

You are an assistant that assesses my progress of discovering this environment and provides useful guidance.
You are required to evaluate if I have met the task requirements. Exceeding the task requirements is also considered a success while failing to meet them requires you to provide critique to help me improve.
IMPORTANT: ONLY SET 'success' TO True IF THE TASK IS DONE AND HAS BEEN ACCOMPLISHED BY THE AGENT. If the last action helped bring the agent closer to succeeding the task, success should still be set to false as the task is not yet completed.
I will give you the following information:
Task: The objective I need to accomplish.
Last action: The last action I performed in the environment.
Context: The context of the task.
Communication: Messages sent by other agents in the environment.
Error: Optionally, you may receive an error message resulting from the previously attempted action. If error is 'None', then no error happened.
Held item: the name of the item currently being held.
An image frame displaying what I can currently see.
If you receive an error you MUST set "success" to false. Critique why you think the error happened and what I should do to overcome it.
IMPORTANT You should only respond in JSON format as described below:
{{ *"reasoning": reasoning, "success": boolean, "critique": critique,* }}
Ensure the response can be parsed by Python 'json.loads', e.g.: no trailing commas, no single

quotes, etc.
Here are some examples: INPUT:
Task: Open the fridge and look inside Last action: OpenObject Fridge Context: ...
Error: None
RESPONSE:
{{ "reasoning": "The fridge door is open therefore the agent successfully opened the fridge", "success": true, "critique": "" }}
INPUT:
Task: Open the fridge and look inside Last action: OpenObject Refrigerator Context: ...
Error: ValueError('No object found with name: Refrigerator')
RESPONSE:
{{ "reasoning": "The environment did not find any objects called 'Refrigerator'. However, there is a fridge in the image so it might be called something different.", "success": false, "critique": "Try using 'Fridge' instead of 'Refrigerator'. " }}
INPUT:
Task: Pick up the egg Last action: PickupObject Egg Context: ...
Error: None Held item: Egg
RESPONSE:
{{ "reasoning": "The egg is being held meaning we picked it up", "success": true, "critique": "" }}
INPUT:
Task: Cook the egg Last action: PickupObject Egg Context: ...
Error: None
RESPONSE:
{{ "reasoning": "The egg is not yet cooked but you are getting closer to achieving that goal by picking it up first", "success": false, "critique": "Now that you have picked up the egg, you can try and find somewhere to put it where you can cook it" }}
INPUT:
Task: Put down egg on counter Last action: PutObject Egg Context: ...
Error: None Held item: Egg
RESPONSE:
{{ "reasoning": "The egg is still held so it was not set down", "success": false, "critique": "Try PutObject Counter to place the egg on the counter" }}
Task: Drop egg Last action: PutObject Egg Context: ...
Error: None Held item: Egg
RESPONSE:
{{ "reasoning": "The egg is still held so it was not set down", "success": false, "critique": "Might need a place to put the egg down on, try PutObject Counter" }}
Task: **{task}**
Last action: **{last_action}**
Context: **{context}**
Communication : **{communication}**
Error: **{error}**
Held item: **{picked_object}**

A.1.3 Skill Memory

The skill memory is tasked with saving learned skills for reuse or to serve as templates for future skills. The inputs used to add a skill are: the action that succeeded the task and the agent's thoughts. These are used to create a skill description through the following prompt sent to an LLM:

Skill Memory, skill description generation

You are a helpful assistant that writes a name and description for a given action command made by another agent. You will be given the action the agent made as well as their thoughts.
1) Do not mention the actual action command
2) Try to summarize the function in no more than 3 sentences.
3) Your response should be a single line of text.
IMPORTANT You should only respond in JSON format as described below:

{{"name": "action name", "description": "description of the action"}}
For example, if the action command is:
SliceObject Bread
Then you would write:
{{"name": "slicing bread", "description": "The agent will slice the bread into bread slices"}}
Please generate a name and a description for the following action:
Agent thoughts: {**agent_thoughts**}
Agent action: {**action**}
EX Response: {{"name": "Holding an egg", "description": "picking up an egg and holding it ready to interact with"}}

This generated skill description is then saved in a vector database[1] with the corresponding generated action name as id. The action is then stored in a dictionary using the action name as key. If a duplicate skill already exists, the old skill is replaced with the new one.

When a skill needs to be fetched, a query is constructed so as to return only the 5 most relevant skills. Fetching relevant skills takes the current task and the current image frame as input. The query is generated by calling an LLM with the following prompt and current image frame:

Skill Memory, query generation

Write a description of the image provided, focusing on the prominent objects in the scene and especially anything that might be relevant to the task at hand. The description should be concise and informative, 3 scentences maximum. Highlighting key details that could assist in understanding the context of the image.
Task: {**task**}.
You should also provide a name for you description.
USE STRICT JSON FORMATTING AS SHOWN IN EXAMPLE BELOW.
EX RESPONSE FORMAT:
{{"name": "...", "description": "...."}}

The description returned by the LLM is then used as a query to search for the 5 most semantically similar skill descriptions in the vector database, as well as their associated unique skill name. The relevant actions that makes up the skills are then fetched from the dictionary using the skill names. These actions are then returned to the agent to use as examples for choosing its next action.

A separate description is stored in the vector database instead of the action itself in order to increase the potential overlap between skills. If the current task requires cooking a potato, a skill describing how to cook and egg might be more useful for than one indicating how to put a potato in the cabinet, even if the latter is semantically closer in terms of action command. For this reason we use skill descriptions to measure semantic distance in the database and thus relevance to the agents current situation.

A.2 Mindforge Modules

The 'Mindforge' agent used in this paper follows the same architecture as the one put forwards by Mindforge [4]. Predforge and Causalforge agents also use the Mindforge architecture, building off it with their respective causal world model enhancements. Mindforge extends Voyager's architecture by adding communication, an episodic memory and a belief system. These then provide further information to the Action Selection moduleA.3 for choosing its next action. The communication module transmits partner agent communication without any extra LLM calls, thus it is not listed below.

A.2.1 Episodic Memory

As opposed to the skill memoryA.1.3, the episodic memory saves an episode each iteration, whether the task was a success or not. An episode consists of: the current task, task beliefs, the agent's previous thoughts, the previous action, whether the action was a success, and the critique generated by the criticA.1.2. The episode itself is semantically rich enough to be saved straight into the vector database. Episodes are queried using the same query generated for the skill memory A.1.3. Before being returned, however, the top 5 episode are first summarized together by an LLM so as to only keep the useful information from these episodes. The summarization prompt is as follows:

[1] https://github.com/chroma-core/chroma

Episodic Memory, episode summarization

You are a helpful assistant tasked with summarizing past episodes and pointing out the causes of failure. Create a concise summary.
Respond in JSON format with field "summary". EX: {{*"summary": "..."*}}
Episodes: **{episodes}**

The generated summary is then sent to the agent. The main goal of the episodic memory is to remember what went wrong in past iterations so the agent does not repeat the same mistakes. Episodes can also capture useful nuances which caused success that the actions returned by the skill memory do not fully explain.

A.2.2 Belief Manager

Mindforge agents hold four different types of beliefs, following the Causal Mind template put forward by Gandhi et al. [5], which we have merged into one class for modularity. These four beliefs are: perception beliefs about the environment it is currently perceiving, partner beliefs describing the other agents in the environment, interaction beliefs which holds information acquired through communication, and task beliefs which details ways in which the agent thinks the task can be achieved.

Firstly, the perception beliefs are created each round straight from the environment observation, which includes the current image frame, any communication recieved from other agents, and any execution errors resulting from the last action. All this information is sent to the LLM through the following prompt:

Belief Manager, perception beliefs

You are a robot exploring an environment with other agents. Provide a set of beliefs that encapsulate the robot's perception of the environment.
You will be given an image frame of the environment, recent communications with other agents and possible execution errors.
Include information about notable items you can see nearby, interactable objects, execution errors and important chat logs.
Communications: **{communications}**
Execution errors: **{error}**
IMPORTANT you MUST respond in the following JSON object format. EX: {{*"beliefs": "..."*}}

The agent holds a separate partner belief for each partner agent in the environment. Partner beliefs are empty during initialization and get updated every iteration. Updating partner beliefs uses previous partner beliefs and any new communication received from that partner. The update is done by an LLM given the following prompt:

Belief Manager, partner beliefs

You are a robot exploring an environment with other agents.
You just had a conversation with another agent based on a task you are trying to solve.
Based on the contents of the conversation and the previous beliefs, you have to create a set of beliefs that represent your perception of the other agent.
The new partner beliefs should contain your perception of the other agent based on the conversation and your previous beliefs. Beliefs should be informative of the other's agent state.
Previous Partner Beliefs: **{previous_partner_belief}**
Conversation: **{conversation}**
IMPORTANT you MUST respond in the following JSON object format. EX: {{*"beliefs": "..."*}}

In the Mindforge paper[4], interaction beliefs are generated whenever a conversation event occurs with partner agents. In our specific setup, agents are continuously communicating, meaning that interaction beliefs are constantly updated each iteration, just like partner beliefs. While partner beliefs are focused on tracking the attributes of other agents, interaction beliefs store any useful information learned through communication that pertains to navigating the environment and solving tasks. This process takes the

current task, communication received, and the previous interaction beliefs as input, and updated by an LLM using the following prompt:

Belief Manager, interaction beliefs

You are a robot exploring an environment with other agents.
You just had a conversation with another agent based on a task you are trying to solve.
Based on the contents of the conversation and the previous beliefs, you have to create a set of beliefs that that can help you complete the task.
The new interaction beliefs should encapsulate useful information from the conversation that can help you complete the task given the conversation and your previous beliefs.
Aim to create a maximum of 5 beliefs. Beliefs should be concise and relevant to the task.
Conversations: {**conversations**}
Previous beliefs: {**previous_interaction_beliefs**}
Current task: {**task**}
IMPORTANT you MUST respond in the following JSON object format. EX: {{*"beliefs": "Belief 1. Belief 2. Belief 3."*}}
Write the each belief as a sentence in the same string. Do not make a list

Task beliefs are initialized every time a new task is selected. They are updated using the current task, the current interaction beliefs and the previous task beliefs by an LLM given the following prompt:

Belief Manager, task beliefs

Create new task beliefs based on the task and the interaction beliefs. If necessary, update the task beliefs to include the new information.
If the previous task beliefs and interaction beliefs contradict, assume the interaction beliefs are true.
Keep the new task beliefs concise and informative.
Previous task beliefs: {**previous_task_beliefs**}
Current task: {**task**}
Interaction beliefs: {**interaction_beliefs**}
IMPORTANT you MUST respond in the following JSON object format. EX: {{*"beliefs": "..."*}}

The updated task beliefs are then passed back to the agent. When a task is succeeded, the task beliefs are saved in a third vector database. These are then fetched whenever new task beliefs needs to be initialized.

A.3 Action Selection

All the information produced by the modules are then used to select the next action to perform in the environment. Depending on what environment the agent will interact with, an environment prompt is also sent to the agent outlining the information required for the agent to interact with it.

A.3.1 Environment prompt

The following is a prompt sent by the environment interface outlining a details about the current environment the agent will be exploring. This section was taken out of the original action selection prompt of Voyager[3], adapted to our environment, and placed as a part of the environment interface to allow for modular swapping of different environments within our framework. This way, swapping to a different environment would also swap the environment prompt, giving the agent new rules on how to interact with that specific environment. The environment prompt for our reduced AI2Thor is as follows:

Action Selection, AI2Thor environment prompt

You cannot move and can only interact with whatever is in front of you.
Choose between the following actions: ["PickupObject", "PutObject", "ToggleObject", "SliceObject", "OpenObject", "NoOp"]
All actions MUST be followed by an object name except for "NoOp". Example: {{*"PickupObject Knife" or "PutObject Pan"*}}

A.3.2 Action selection

The next action the agent performs in the environment is determined by an LLM call. The LLM in the Voyager agent receives a specific set of inputs: the current image frame, communication, the environment prompt, the current task, the last action, the critique generated by the critic A.1.2, any errors received from the environment, retrieved skills A.1.3, and the object currently being held. Mindforge agents receive these inputs plus a summary of past episodes A.2.1 and all four beliefs. The prompt designed to package this comprehensive information and instruct the LLM on how to select an action is as follows:

Action Selection, main prompt

You are controlling a robot in an environment with other agents. Every round you must select actions for it to perform in the environment. Your goal is to explore the environment and communicate with other agents.
ONLY Respond STRICTLY in valid JSON format with 'thoughts', 'action' and 'communication' fields.
{environment_prompt}.
Example full response: {{*"thoughts": "I see a kitchen counter", "action": "ToggleObject Stove", "communication": "I am turning on the stove"*}}
Each round I will give you:
Task: The task you should try to achieve
Last action: The action performed last round
Held item: Item currently being held
Critique: A proposal of what you could do differently to achieve the task
Error: An error message returned by the environment. None if no error was returned
Relevant skills: Descriptions of passed skills that succeeded and the exact command used to do them
Past episodes: A summary of relevant past episodes
Task belief: Beliefs about the current task
Perception belief: Beliefs about the current environment around you
Interaction beliefs: Beliefs based about what the other agents have told you
Partner beliefs: Beliefs about your partner agents
Communications from other agents: ...
An image of what you can currently see in-front of you.
Example response format: {{*"thoughts": "Description of what you see and what you want to do", "action": "action to execute in the environment", "communication": "Send message to other agents"*}}
Task: **{task}**
Last action: **{last_action}**
Held item: **{picked_object}**
Critique: **{critique}**
Error: **{error}**
Relevant skills: **{skill_memory}**
Past episodes: **{episode_summary}**
Task belief: **{task_beliefs}**
Perception belief: **{perception_beliefs}**
Interaction beliefs: **{interaction_beliefs}**
Partner beliefs: **{partner_beliefs}**
Communications from other agents: **{formatted_communication}**.

For both the Voyager and Mindforge agents, the generated action is executed directly in the environment. However, for the Predforge agent, the action is first sent to the Causal World Model, which generates a prediction of the effect the action will have in the environment. This action and its prediction are then sent back to the action selection module, which is tasked with choosing an action again, but this time it is equipped with the added information. The following is thus appended to the main prompt with the predicted image frame:

Action Selection, with prediction

I have also provided a prediction of what would happen if you chose to perform action

"**{pred_action}**" in the environment.
You can use this prediction to see if you want to perform that action, or you would rather perform another action.
THE FOLLOWING IMAGE HAS NOT ALREADY HAPPENED IN THE ENVIRONMENT. You would need to first select action '**{pred_action}**' The prediction is passed as an image below:

A.3.3 Candidate action selection

The final function of the action selection module is to generate a set of candidate actions that are then passed to the surgical intervention module. The prompt used for this task is very similar to the main action prompt, except the LLM is specifically instructed to select candidates that it believes would influence the originally chosen action. This crucial step filters out irrelevant actions that would have a low chance of being valid interventions, thereby saving time and compute.

You are controlling a robot in an environment with other agents. I will give you the action the robot is intending to perform. Give me a set list of 3-5 other candidate actions the robot could perform that would impact the result of the initial action.
ONLY Respond STRICTLY in valid JSON format with 'candidate_actions' field.
{environment_prompt}.
Example full response: {{*"candidate_actions": ["PickupObject Egg", "ToggleObject Microwave", "OpenObject Cabinet"]*}}
Each round I will give you:
Task: The task you should try to achieve
Last action: The action performed last round
Held item: Item currently being held
Critique: A proposal of what you could do differently to achieve the task
Error: An error message returned by the environment. None if no error was returned
Relevant skills: Descriptions of passed skills that succeeded and the exact command used to do them
Past episodes: A summary of relevant past episodes
Task belief: Beliefs about the current task
Perception belief: Beliefs about the current environment around you
Interaction beliefs: Beliefs based about what the other agents have told you
Partner beliefs: Beliefs about your partner agents
Communications from other agents: ...
An image of what you can currently see infront of you.
Example response format: {{*"candidate_actions": ["ToggleObject Stove_One", "PickupObject Plate", "OpenObject Microwave"]*}}
Task: **{task}**
Last action: **{last_action}**
Held item: **{picked_object}**
Critique: **{critique}**
Error: **{error}**
Relevant skills: **{skill_memory}**
Past episodes: **{episode_summary}**
Task belief: **{task_beliefs}**
Perception belief: **{perception_beliefs}**
Interaction beliefs: **{interaction_beliefs}**
Partner beliefs: **{partner_beliefs}**
Envisioned action: **{selected_action}**

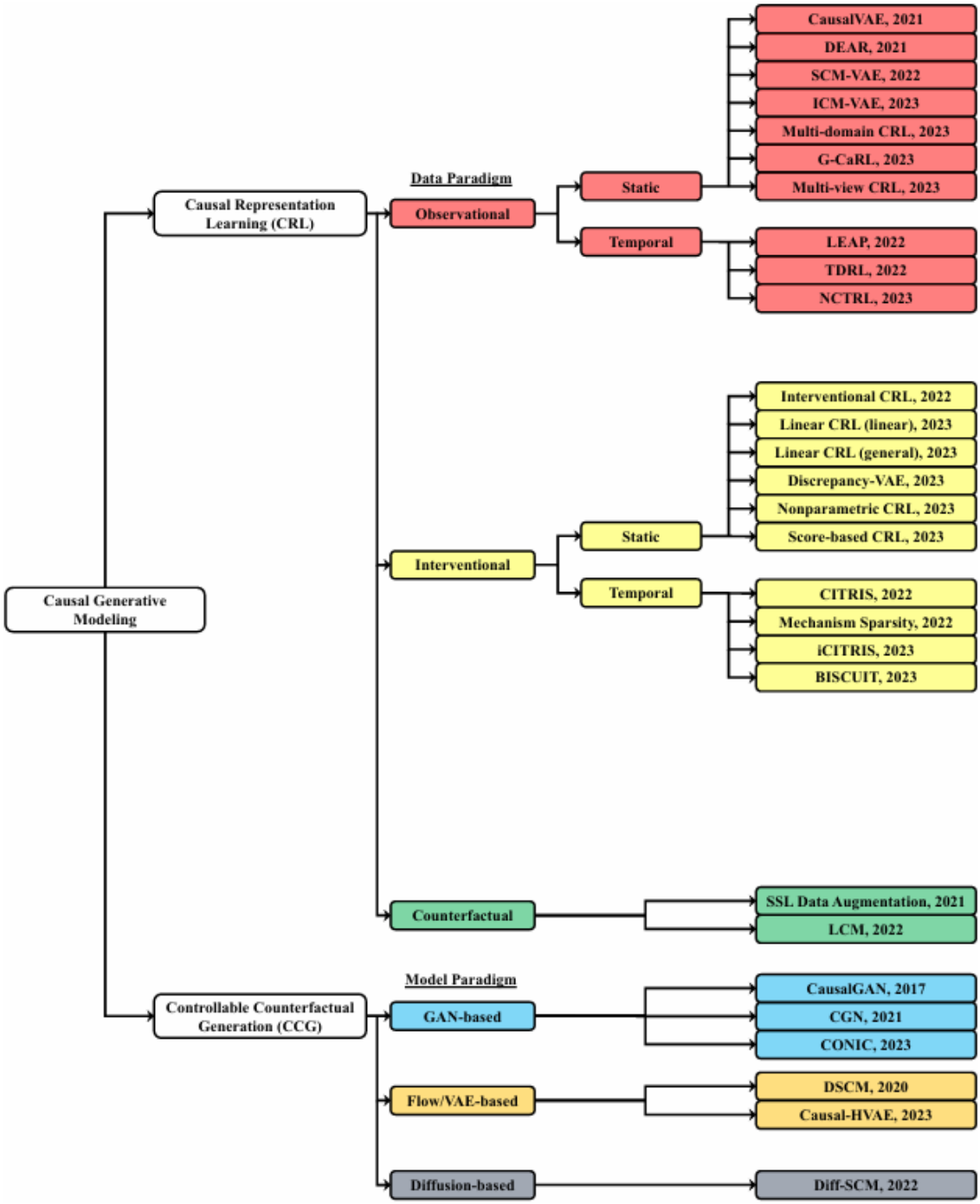

Figure 16: Presented is a taxonomy of the different types of Causal Generative Modeling put forwards by Komanduri et al. [6]. We have chosen BISCUIT[18] for our agent as it works with interventional data in a temporal environment. The other options in this category are either precursors to BISCUIT (CITRIS[26] and iCITRIS[27]), or use a sparsity constraint not useful in our environment (Mechanism Sparsity[19])